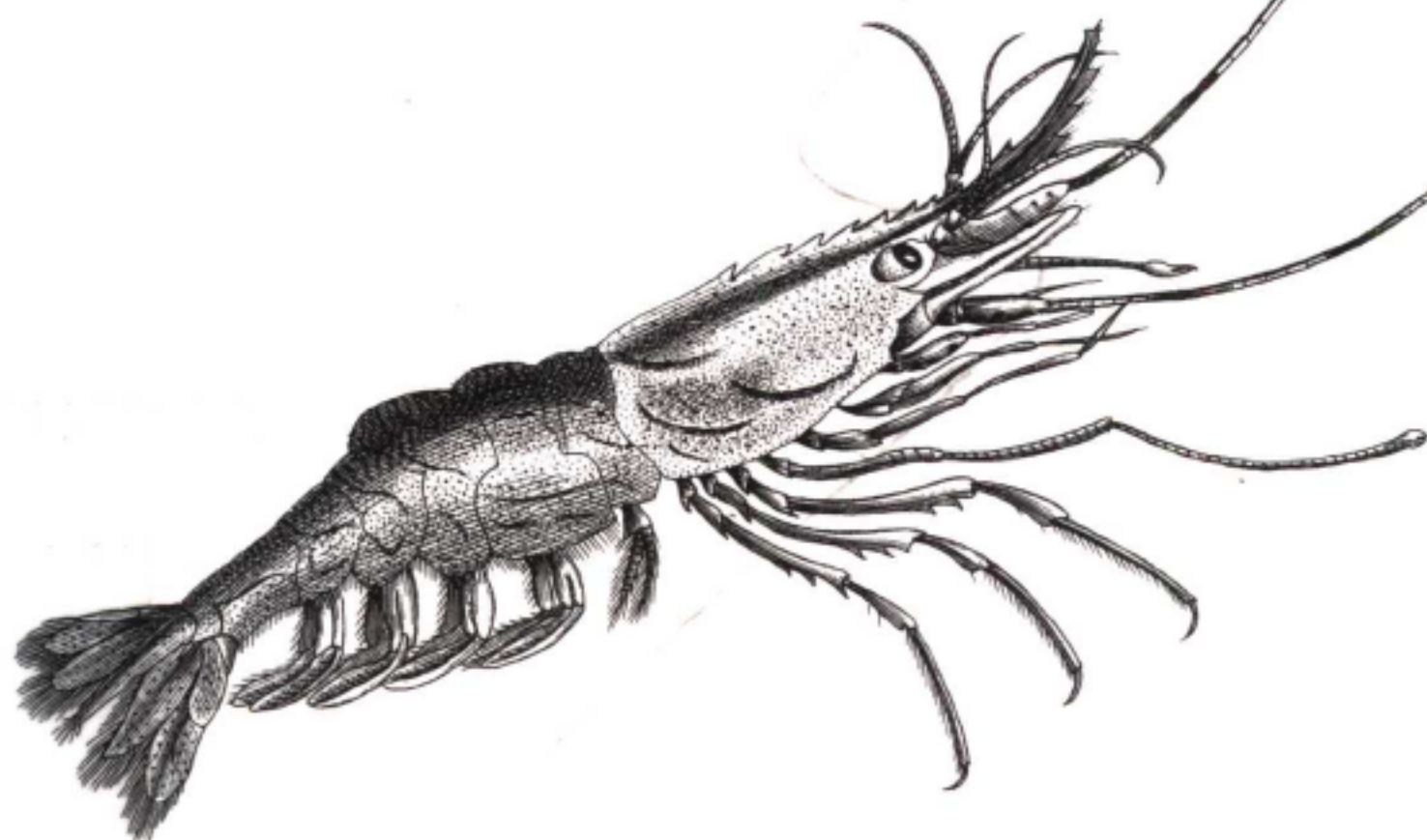


Eclipse Cookbook

涵蓋
Eclipse 3.0

Eclipse Cookbook™ 中文版



Steve Holzner 著
王欣轩 译

O'REILLY®



清华大学出版社

Eclipse Cookbook 中文版



Eclipse 是一个集成开发环境 (IDE)，它将代码编辑器、编译器、调试器、文本编辑器、GUI 生成器和其他组件集成到单个用户友好的应用程序中。Eclipse 提供了一个扎实的基础，使 Java 开发人员能够构建和运行集成的软件开发工具，进行 Web 开发、应用程序设计、建模、实现、测试等。

针对使用 Eclipse 这个新的 Java 开发平台时可能遇到的 175 种情况——从自动部署 Web 应用程序，到逆向工程编译代码，从跨越多个包重命名类的所有引用，到初始化 SWT JNI 库——本书提供了面向任务的解决方案。

本书汇集了针对复杂的 IDE 日常任务的大量解决方案，例如：

- 在多种环境下安装和设置 Eclipse
- 自动代码填充和自定义组合键
- 设置 Eclipse，以自动添加标记和纠正语法错误
- 自动扩展和实现接口
- 使用 JUnit 进行自动测试和集成测试
- 将 Eclipse 连接到 CVS 存储库
- 使用 Ant 自动编译大型项目，包括应用程序的运行和部署
- 广泛介绍 SWT，包括非矩形窗口、标签文件夹和浏览器等新特性
- 在 SWT 应用程序中嵌入 Swing 和 AWT 窗口
- 使用 Eclipse 插件框架创建菜单和透视图
- 创建插件向导和视图

各节采用 Cookbook 系列特有的“问题/解决方案/讨论”模式对问题进行了清晰、透彻的描述，简要而完整地讨论了解决方案，并举例说明了解决方案。本书可以满足各个层次的 Java 程序员的需要，尤其是那些打算超越教程（不仅仅是编写插件和扩展）和每天使用 Eclipse 的程序员。

ISBN 978-7-302-14499-1



9 787302 144991 >

定价：42.00元

Eclipse CookbookTM

中文版

Steve Holzner 著

王欣轩 译

O'REILLY[®]

Beijing • Cambridge • Farnham • Köln • Paris • Sebastopol • Taipei • Tokyo

O'Reilly Media, Inc. 授权清华大学出版社出版

清华大学出版社

Copyright ©2004 by O'Reilly Media, Inc.

Authorized Simplified Chinese translation edition, by O'Reilly Media, Inc., is published by Tsinghua University Press, 2007. Authorized translation of the original English edition, 2004 O'Reilly Media, Inc., the owner of all rights to publish and sell the same.

All rights reserved including the rights of reproduction in whole or in part in any form.

本书之英文原版由 O'Reilly Media, Inc. 于 2004 年出版。

本中文简体翻译版由 O'Reilly Media, Inc. 授权清华大学出版社于 2007 年出版。此翻译版的出版和销售得到出版权和销售权的所有者——O'Reilly Media, Inc. 的许可。

版权所有，未经书面许可，本书的任何部分和全部不得以任何形式复制。

北京市版权局著作权合同登记

图字：01-2006-7114 号

本书封面贴有清华大学出版社防伪标签，无标签者不得销售。

版权所有，侵权必究。侵权举报电话：010-62782989 13501256678 13801310933

图书在版编目 (CIP) 数据

Eclipse Cookbook™ 中文版 / (美) 霍兹纳 (Holzner, S.) 著；王欣轩译．—北京：清华大学出版社，2007.2

书名原文：Eclipse Cookbook

ISBN 978-7-302-14499-1

I. E… II. ①霍… ②王… III. 软件工具—程序设计 IV. TP311.56

中国版本图书馆 CIP 数据核字 (2007) 第 003794 号

责任编辑：常晓波

封面设计：Emma Cotby, 张 健

责任校对：张 剑

责任印制：王秀菊

出版发行：清华大学出版社

<http://www.tup.com.cn>

c-service@tup.tsinghua.edu.cn

社 总 机：010-62770175

投稿咨询：010-62772015

地 址：北京清华大学学研大厦 A 座

邮 编：100084

邮购热线：010-62786544

客户服务：010-62776969

印 刷 者：北京市清华园胶印厂

装 订 者：三河市李旗庄少明装订厂

经 销：全国新华书店

开 本：178×233 印张：23

版 次：2007 年 2 月第 1 版

印 数：1~3000 册

定 价：42.00 元(册)

字数：497 千字

印次：2007 年 2 月第 1 次印刷

本书如存在文字不清、漏印、缺页、倒页、脱页等印装质量问题，请与清华大学出版社出版部联系调换。

联系电话：010-62770177 转 3103 产品编号：023388-01

TP311.56
319

2007

Eclipse CookbookTM

中文版

作者简介

Steve Holzner 是一名获奖作者，自 Java 面世以来，他一直在从事 Java 题材方面的写作。他曾担任 PC Magazine 的特约编辑，其许多著作被翻译成 18 种语言，读者遍及全世界。他编写的书已经销售了超过 150 万册，其中许多畅销书都是关于 Java 的。Steve 还为 O'Reilly 出版公司编写了 Eclipse 一书。

Steve 从麻省理工学院 (MIT) 毕业，在康奈尔大学获得博士学位；在麻省理工学院 (MIT) 和康奈尔大学他都是最受学生欢迎的教员之一；在过去的几年里他教授过数千名学生，学生给他的平均评价为 4.9 分 (满分是 5.0 分)。他还经营一家自己的软件公司，面向全美的企业程序设计人员开设多门为期一周的 Java 课程。

封面介绍

本书封面上的动物是一种长臂虾 (罗氏沼虾, *Macrobrachium rosenbergii*)。这种甲壳类动物生长于淡水河流与湖泊中,也见于印度洋-太平洋地区的咸淡水水域与河流的入海口处。

这种虾外形与小龙虾相似,但长有两只长臂,长臂上长满细爪。与大多数节肢类动物类似,长臂虾有一副外骨骼;其肌肉不是通过内骨骼相连,而是连接到腹部的、硬硬的、富钙的甲壳上。六根长长的触须上布满化学感受器细胞,使长臂虾能够探测到水中的食物的气味。长臂虾用这些触须不停地相互擦洗,以清除影响其触觉的污物。

像所有甲壳类动物一样,长臂虾必须蜕皮才能生长。在蜕皮之前,旧的壳下面长出一个新壳;新壳是软的,而且可以适当弯曲,像一个空的气球。新的壳通过充水进行膨胀,在旧壳的薄弱处将其撕裂。旧壳撕裂成两半,整个头部破壳而出。尾部快速轻弹,将旧的外骨骼留在了海底。

O'Reilly Media, Inc. 介绍

为了满足读者对网络 and 软件技术知识的迫切需求，世界著名计算机图书出版机构 O'Reilly Media, Inc. 授权清华大学出版社，翻译出版一批该公司久负盛名的英文经典技术专著。

O'Reilly Media, Inc. 是世界上在 Unix、X、Internet 和其他开放系统图书领域具有领导地位的出版公司，同时也是联机出版的先锋。

从最畅销的 *The Whole Internet User's Guide & Catalog* (被纽约公共图书馆评为 20 世纪最重要的 50 本书之一) 到 GNN (最早的 Internet 门户和商业网站)，再到 WebSite (第一个桌面 PC 的 Web 服务器软件)，O'Reilly Media, Inc. 一直处于 Internet 发展的最前沿。

许多书店的反馈表明，O'Reilly Media, Inc. 是最稳定的计算机图书出版商——每一本书都一版再版。与大多数计算机图书出版商相比，O'Reilly Media, Inc. 具有深厚的计算机专业背景，这使得 O'Reilly Media, Inc. 形成了一个非常不同于其他出版商的出版方针。O'Reilly Media, Inc. 所有的编辑人员以前都是程序员，或者是顶尖级的技术专家。O'Reilly Media, Inc. 还有许多固定的作者群体——他们本身是相关领域的技术专家、咨询专家，而现在编写著作，O'Reilly Media, Inc. 依靠他们及时地推出图书。因为 O'Reilly Media, Inc. 紧密地与计算机业界联系着，所以 O'Reilly Media, Inc. 知道市场上真正需要什么图书。

目录

前言	1
第 1 章 基本技能	7
1.0 简介	7
1.1 获得 Eclipse	7
1.2 安装和运行 Eclipse	9
1.3 理解 Eclipse 工作区	11
1.4 运行多个 Eclipse 窗口	14
1.5 创建一个 Java 项目	16
1.6 管理透视图、视图和编辑器	18
1.7 掌握 Java 透视图	20
1.8 创建一个 Java 类	22
1.9 自动编写代码	24
1.10 运行代码	28
1.11 运行代码片段	29
1.12 自动修复语法错误	31
1.13 保持工作区的整洁	34
1.14 从灾难中恢复	36

第 2 章 使用 Eclipse	37
2.0 简介	37
2.1 显示或隐藏视图	38
2.2 移动视图或工具栏	39
2.3 访问任何项目文件	41
2.4 平铺编辑器	41
2.5 最大化视图和编辑器	44
2.6 返回上一个编辑器	45
2.7 返回到上一个编辑位置	45
2.8 将视图链接至编辑器	46
2.9 重新排序视图和编辑器标签	47
2.10 从编辑器导航到视图	47
2.11 指定组合键	48
2.12 通过图标显示更多的资源信息	49
2.13 使用不同的工作区	50
2.14 创建任务	51
2.15 创建书签	52
2.16 创建快速视图	53
2.17 自定义帮助	55
2.18 恢复删除的资源	56
2.19 自定义透视图	57
2.20 还原透视图	58
2.21 创建新的透视图	59
 第 3 章 Java 开发	 61
3.0 简介	61
3.1 加速 JDT 编辑器	61
3.2 创建一个 Java 项目	63
3.3 创建 Java 包	65
3.4 创建 Java 类	67

3.5	创建 Java 方法	68
3.6	覆盖 Java 方法	70
3.7	获取方法参数的提示	71
3.8	插入方法参数名	72
3.9	创建 getter/setter 方法	73
3.10	创建委托方法	74
3.11	用 do/for/if/try/while 块包围代码	75
3.12	查找匹配的花括号	76
3.13	自动包围字符串	77
3.14	创建构造函数	77
3.15	将构造函数转换为工厂方法	79
3.16	注释掉一段代码	81
3.17	创建工作集	81
3.18	创建 TODO 任务	84
3.19	自定义代码助手	85
 第 4 章 重构、编译和运行		89
4.0	简介	89
4.1	重命名元素	90
4.2	移动元素	93
4.3	接口的提取与实现	95
4.4	搜索代码	97
4.5	比较文件	101
4.6	根据本地历史记录比较文件	102
4.7	从本地历史记录恢复元素和文件	104
4.8	选择 Java 编译运行环境	105
4.9	运行代码	107
4.10	编译代码	108
4.11	使用 .jar 和 .class 文件	110
4.12	设置运行配置	113

第 5 章 测试和调试	116
5.0 简介	116
5.1 安装 JUnit	117
5.2 用 JUnit 测试应用程序	119
5.3 启动调试会话	125
5.4 设置断点	128
5.5 单步调试代码	131
5.6 在遇到断点前继续运行	133
5.7 运行选定的代码行	136
5.8 监视表达式和变量	136
5.9 为断点设置命中计数	137
5.10 配置断点条件	139
5.11 创建字段、方法和异常断点	140
5.12 计算表达式的值	143
5.13 在调试期间为变量赋值	146
5.14 快速修改代码	147
第 6 章 使用 Eclipse 进行团队开发	149
6.0 简介	149
6.1 获得 CVS 服务器	150
6.2 创建 CVS 储存库	151
6.3 将 Eclipse 连接至 CVS 储存库	152
6.4 将 Eclipse 项目储存在 CVS 储存库中	155
6.5 将文件提交到 CVS 储存库中	157
6.6 以可视化方式标记版本控制下的文件	158
6.7 检查 CVS 储存库	159
6.8 从 CVS 储存库检出项目	161
6.9 从 CVS 储存库更新本地代码	162
6.10 将你的代码与 CVS 储存库同步	164
6.11 创建代码补丁	166

6.12 为代码版本命名	168
6.13 创建 CVS 分支	170

第 7 章 Eclipse 和 Ant..... 175

7.0 简介	175
7.1 将 Ant 连接至 Eclipse	175
7.2 使用 Ant 编译 Eclipse 应用程序	181
7.3 捕获 Ant 编译文件语法问题	184
7.4 使用不同的编译文件	186
7.5 使用你自己的 Ant 版本	188
7.6 设置类型和全局属性	190
7.7 设置 Ant 编辑器的选项	191
7.8 设置 Ant 参数	191
7.9 使用 Ant 视图	194
7.10 将 Ant 用作外部工具	195

第 8 章 SWT：文本、按钮、列表和非矩形窗口 196

8.0 简介	196
8.1 使用 SWT 窗口小部件	199
8.2 创建一个 SWT 应用程序	203
8.3 将所需的 SWT JAR 文件添加到编译路径中	206
8.4 启动 SWT 应用程序	207
8.5 定位窗口小部件和使用布局	210
8.6 创建按钮和文本小部件	212
8.7 处理 SWT 窗口小部件事件	214
8.8 创建列表小部件	217
8.9 创建复合小部件	221
8.10 创建非矩形窗口	222
8.11 多线程 SWT 应用程序	225

第 9 章 SWT：对话框、工具栏及菜单等	226
9.0 简介	226
9.1 创建消息框	226
9.2 创建对话框	227
9.3 创建工具栏	232
9.4 在工具栏上嵌入按钮	233
9.5 处理工具栏事件	234
9.6 在工具栏上嵌入组合框、文本小部件和菜单	236
9.7 创建菜单系统	237
9.8 创建文本菜单项	240
9.9 创建图像菜单项	244
9.10 创建单选菜单项	244
9.11 创建菜单项加速键和助记符	246
9.12 启用和禁用菜单项	246
9.13 创建菜单分隔符	247
9.14 创建表格	247
9.15 创建表格列	251
9.16 为表格项添加复选标记	252
9.17 启用和禁用表格项	253
9.18 为表格项添加图像	254
9.19 在 SWT 内部使用 Swing 和 AWT	255
第 10 章 SWT：控件工具栏、标签文件夹、 树和浏览器	258
10.0 简介	258
10.1 创建 SWT 标签文件夹	258
10.2 创建 SWT 控件工具栏	261
10.3 在控件工具栏上添加控件项	262
10.4 在控件工具栏上添加下拉菜单	265
10.5 创建 SWT 树	270

10.6	处理树事件	273
10.7	在树项中添加复选框	275
10.8	在树项中添加图像	277
10.9	创建 SWT 浏览器小部件	277
 第 11 章 JSP、Servlet 和 Eclipse		282
11.0	简介	282
11.1	安装 Tomcat	282
11.2	启动 Tomcat	284
11.3	创建 JSP 文件	285
11.4	创建 Servlet	288
11.5	在 Tomcat 中安装 Servlet	289
11.6	就地创建 Servlet	292
11.7	就地编辑 web.xml 文件	294
11.8	避免输出文件夹被擦除	296
11.9	连接到 Java 组件	297
11.10	使用 Tomcat 插件	299
11.11	创建 WAR 文件	303
 第 12 章 创建插件：扩展点、动作和菜单		307
12.0	简介	307
12.1	安装插件	307
12.2	创建 plugin.xml 文件	309
12.3	使用向导创建基于菜单的插件	310
12.4	使用运行时工作台测试插件	315
12.5	部署插件	318
12.6	使用框架编写插件	319
12.7	在插件中响应用户的动作	322
12.8	从头创建插件菜单	324
12.9	创建动作	327

12.10 为插件动作编写代码	328
12.11 将插件自动添加到透视图中	331
第 13 章 创建插件：向导、编辑器和视图	333
13.0 简介	333
13.1 创建支持向导和编辑器的插件	333
13.2 自定义向导	337
13.3 自定义编辑器	340
13.4 创建支持视图的插件	343
13.5 在视图添加控件	347
13.6 配置视图的动作	348

前言

本书可以帮助读者全面了解 Eclipse，这个当今最流行的 Java 集成开发环境（IDE）。Eclipse 是一个非常好的开发工具，但也是一个比较复杂的工具，并非每一个人都有时间去整天研究它。这就是编写本书的初衷；本书将为你揭开 Eclipse 的神秘面纱。

在 Java 的世界里，Eclipse 一直是不可或缺的。在它之前，已经有不少的 Java IDE，但 Eclipse 却与众不同。如果你一直只使用 Java 命令行编译器 `javac` —— 它根本无法与 Eclipse 相提并论，那么 Eclipse 将使你的编程生涯发生革命性的变化。Eclipse 实在是太好了，因为它就是为处理编程细节而设计的，这些细节包括语法检查、错误处理、添加必要的 `import` 语句、设定编译目标、用一次单击注释掉代码块、把 `.jar` 文件添加到编译路径中、重构、甚至重新设置代码的格式。Eclipse 可以为你完成这一切。然而，在 Eclipse 中起决定作用的实际上是你，你可以很快地掌握 Eclipse。

本书可以缩短学习过程。如果遇到问题，只需查阅相应的章节即可弄清如何解决这个问题。如果需要返回到以前编辑过的位置，可参阅第 2 章。如果想通过几次单击提取代码中的接口，可参阅第 4 章。如果想再多用几次单击就创建 `getter` 和 `setter` 方法，请查阅第 3 章。如果想把 Eclipse 连接到一个 CVS 储存库，请参阅第 6 章。

本书详细介绍了 Eclipse，从基本功能到高级应用。本书的特别设计可以使你快速解决几乎所有关于 Eclipse 的问题，不会浪费你的时间。今天，Eclipse 是 Java 领域的热点，而本书提供关于掌握 Eclipse 所需的一切。

本书内容

本书详细剖析 Eclipse。本书将解决数百个 Eclipse 问题，讨论数十个议题，从安装的全

过程，到崩溃时的重新安装。本书还将介绍 Eclipse 3.0 必需具备的功能。下面概括介绍本书的内容。

第 1 章，基本技能

本章介绍一些基本技能——使用 Eclipse 和处理日常任务所需要掌握的一切基本方法，包括获得和安装 Eclipse。

第 2 章，使用 Eclipse

本章主要介绍 Eclipse 的工作台及其功能。深入介绍了编辑器、视图、透视图等及其使用方法。

第 3 章，Java 开发

Eclipse 的优势在于 Java 开发，而从本章开始，本书将进入 Java 开发。本章将使用 Java 开发工具（JDT）创建和处理 Java 项目、类、方法、代码等。

第 4 章，重构、编译和运行

重构主要处理在代码中重命名或移动元素并更新代码中的所有实例等任务。本章介绍重构以及许多其他的 Java 高级任务。本章还将介绍项目的编译和运行，包括设置运行配置。

第 5 章，测试和调试

如果不能进行调试，IDE 还有何用？Eclipse 的调试器是第一流的，读者可以通过本章全面认识 Eclipse 的调试器，包括断点、断点命中计数器、观察点，从而可以快速修改代码，等等。

第 6 章，使用 Eclipse 进行团队开发

Eclipse 也是为在团队中应用而设计的。本章将介绍如何使用 Eclipse 和并行版本系统（CVS）服务器，以便共享代码。本章还将介绍如何把 Eclipse 连接到 CVS 服务器，如何将 Eclipse 项目存储在 CVS 储存库中，如何查看文件和项目，等等。

第 7 章，Eclipse 和 Ant

Ant 是最好的 Java 编译工具，而 Eclipse 已经内置了对 Ant 的支持。本章介绍如何创建 Ant 编译文件、如何执行编译文件以及在 Eclipse 中使用 Ant 能够做什么。

第 8 章，SWT：文本、按钮、列表和非矩形窗口

Eclipse 中内置了标准窗口小部件工具包（SWT），这是一个应用广泛的 GUI API，专门为取代 Java 的 AWT 和 Swing 而设计。这是关于 SWT 的第一章，内容包括 SWT 的基础知识以及如何开始使用一些基本的窗口小部件，如按钮、列表和复合小部件等，还介绍了如何创建非矩形窗口。

第9章，SWT：对话框、工具栏及菜单等

本章进一步介绍 SWT 窗口小部件，包括一些高级的窗口小部件，如对话框、工具栏、菜单和表格。本章还将介绍如何将 AWT/Swing 窗口嵌入到 SWT 应用程序中。

第10章，SWT：控件工具栏、标签文件夹、树和浏览器

这是关于 SWT 的最后一章，进一步介绍了 SWT 窗口小部件，包括控件工具栏、标签文件夹、树和浏览器。

第11章，JSP、Servlet 和 Eclipse

Eclipse 和 Web 开发是天生的一对。本章介绍如何使用 Eclipse 开发 Web 应用程序，包括 JSP、JavaBeans、Servlet。本章还将介绍如何为 Web 应用程序创建部署包。

第12章，创建插件：扩展点、动作和菜单

本章及下一章介绍如何创建自定义的 Eclipse 插件。本章详细介绍扩展点、动作和创建插件菜单。

第13章，创建插件：向导、编辑器和视图

本章将结束对插件的关注；本章将创建可在 Eclipse 中显示向导、视图和编辑器的插件。

排版约定

下面是本书中采用的体例约定：

纯文本

表示菜单标题、菜单项、菜单按钮和键盘加速键。

斜体 (*Italic*)

表示新术语、URL、电子邮件地址、文件名、文件扩展名、路径名、目录和 Unix 使用程序。

等宽字体 (Constant Width)

表示命令、选项、开关、变量、类型、类、名称空间、方法、模块、属性、参数、值、对象、事件、事件句柄或 XML 标记。

等宽斜体 (*Constant Width Italic*)

表示应该用用户提供的值替换的文本。

本书中还使用了许多其他的约定。例如，菜单项用一个 \rightarrow 符号隔开，如 File \rightarrow New \rightarrow Project。为了突出新加的代码，新加的代码加粗显示；而且用 3 个竖点表示省略了一些代码。示例如下：

```
for (int loopIndex = 0; loopIndex < 10; loopIndex++)
{
    TabItem tabItem = new TabItem(tabFolder, SWT.NULL);
    tabItem.setText("Tab " + loopIndex);

    Text text = new Text(tabFolder, SWT.BORDER);
    text.setText("This is page " + loopIndex);
    tabItem.setControl(text);
    .
    .
    .
}
```

需要的软件

本书中需要的任何软件，包括 Eclipse，都可以从网上免费获得。编写本书时使用的是 Eclipse 2.1.2 以及 Eclipse 3.0 的早期版本。第 1 章介绍从何处获得 Eclipse 以及如何安装和配置 Eclipse。

本书还将介绍从何处可以获得其他软件，从 Tomcat Web 服务器到 CVS 服务器。在第 12 章和第 13 章中创建插件时，就是在使用 Eclipse 开发自己的软件。

读者所需要的所有软件都可以免费获得；只需下载即可。本书将介绍从何处可以获得这些软件。

使用代码示例

本书中开发的所有代码，都可以从本书的 O'Reilly Web 站点免费下载，网址是 <http://www.oreilly.com/catalog/eclipseckbk>（在安装前，请阅读下载的 *readme.txt* 文件）。

本书旨在帮助你完成工作。一般而言，你可以在自己的程序和文档中使用本书中的代码。你不必与我们联系以获得许可，除非你要复制代码的重要部分。例如，编写一个程序，其中使用了本书中的几个代码块，无需取得许可。销售或发行包含 O'Reilly 图书中的示例的 CD-ROM 时，必须取得许可。引用本书及其示例代码来解答问题时，不需要获得许可。若要将本书中的大量示例代码加入你的产品文档，则必须取得许可。

我们感谢但不要求注明出处。在注明出处时，通常应包括书名、作者、出版社和 ISBN。例如：“Eclipse Cookbook, by Steve Holzner. Copyright 2004 O'Reilly, 0-596-00710-8”。

如果你感到对代码示例的使用不在公平使用或上述许可范围之内，方便的时候可通过 permissions@oreilly.com 与我们联系。

我们很想听到你的建议

请通过以下地址把关于本书的评论和问题发送给出版社：

美国：

O'Reilly Media, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472

中国：

100080 北京市海淀区知春路 49 号希格玛公寓 B 座 809 室
奥莱理软件（北京）有限公司

本书的 Web 页上列出了勘误表、示例和任何额外的信息。可登录以下网址查询：

<http://www.oreilly.com/catalog/eclipseckbk>
<http://www.oreilly.com.cn/book.php?bn=978-7-302-14499-1>

如果想就本书的技术问题发表评论或咨询，请发邮件至：

bookquestions@oreilly.com
info@mail.oreilly.com.cn

关于书本、会议、资源中心和 O'Reilly 网络的更多信息，请访问 O'Reilly 的 Web 站点：

<http://www.oreilly.com>
<http://www.oreilly.com.cn>

参加本书翻译的人员有：王欣轩、陈宗斌、蔡京平、李毅、毕蓉蓉、祁海生、张贺乾、史宁、刘绿生、孙雷、蔡加双、安东辉、米翔娟、刘颜、王宇宇、沈程亮、陆晓萍、金国良、俞群、李正智、赵敏、陈征、陈红霞。

基本技能

1.0 简介

如果你是一名 Java 程序员，那么 Eclipse 无疑是最好的工具。在降低 Java 开发的难度方面，Eclipse 不仅好于其他任何开发环境，而且它可以免费下载！

即使使用 Java 编译器 `javac` 进行开发，Eclipse 也能使开发变得更加轻松。事实上，Eclipse 的集成开发环境（IDE）使开发过程变得尽可能有趣：Java 程序员首次启动并开始使用 Eclipse 时，他们通常都会认为，Eclipse 太棒了！

然而，与任何广泛使用的编程工具一样，Eclipse 也有一个学习曲线。在这一章，我们将尽快掌握 Eclipse 的基本技能，从安装 Eclipse，到创建一个简单的 Java 应用程序。其中一些技能仅供参考，你可能已经掌握了它们，而另外一些技能可能是新的。本书的目标是推广 Eclipse；为此，你必须牢固掌握本章介绍的基本技能。

1.1 获得 Eclipse

问题

你想试用一下 Eclipse。

解决方案

Eclipse 是免费的，可以从站点 <http://www.eclipse.org> 下载。只需单击该页面上左侧的 *Downloads* 链接即可。

讨论

当前的下载 URL 是 <http://www.eclipse.org/downloads/index.php>。下载页面如图 1-1 所示。

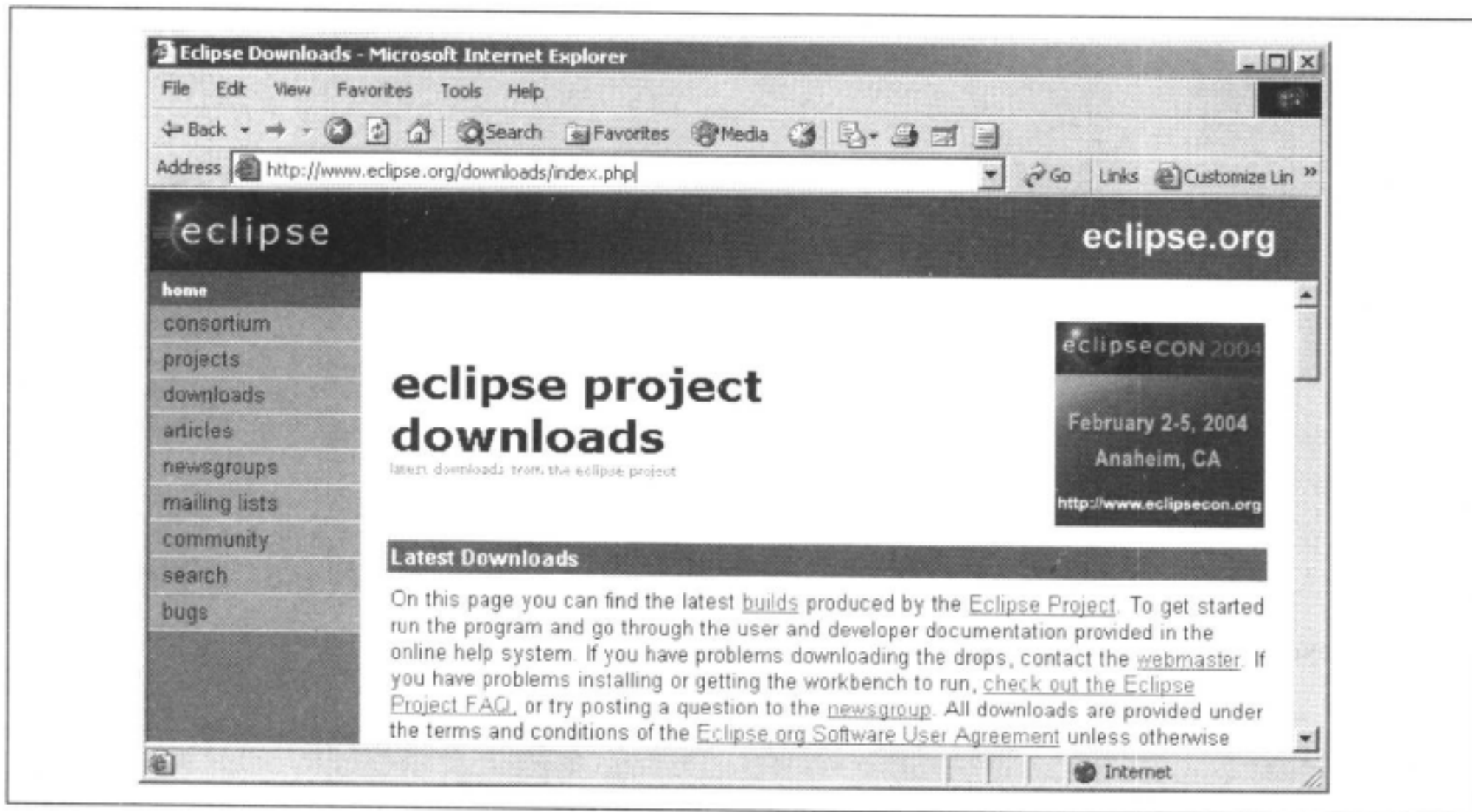


图 1-1: Eclipse 下载页面

单击最近的一个镜像站点，选择所需的 Eclipse 版本，并选择操作系统（Eclipse 适用于所有操作系统，从 Linux 到 AIX，到 Windows）。下载将自动开始。

Eclipse 有 4 种不同的版本可供下载：

发布版

这种版本是通用的。大多数情况下，在下载 Eclipse 时，都应下载这种版本。这种版本已经过了测试，出现严重错误的可能性很小。假如 Eclipse 不是免费的，那么销售的应该是这种版本。

稳定版

稳定版与测试版非常相似。Eclipse 开发小组将这种版本视为相对稳定的版本，但与任何测试版一样，运行时可能会出现问题。如果有兴趣，这种版本是发现 Eclipse 未来新特性的地方。

集成版

这种版本稍逊于稳定版，其组件已经过相当严格的测试，但它们集成在一起运行时仍然可能存在问题。如果集成版运行良好，那么它就升级为稳定版。

每晚版

所有公开提供的Eclipse版本中最具试验性的、风险最大的版本，由Eclipse开发小组每晚创建，不提供任何保证。笔者使用这种版本的经验表明，可能会遇到大量的问题，所以要慎用。

通常，应该下载Eclipse的最新的发布版。下载可能要花一些时间。根据用户使用的操作系统的不同，多数最新的发布版均超过60 MB。

参考

Eclipse (O'Reilly) 一书的第1章；Eclipse 站点 <http://www.eclipse.org>；站点 www.eclipse.org/articles/index.html 的技术文章；站点 <http://www.eclipse.org/newsgroups/index.html> 的新闻组；站点 <http://www.eclipse.org/eclipse/development/main.html> 的当前版本和未来版本页。

1.2 安装和运行 Eclipse

问题

你想在自己的机器上安装并运行Eclipse。

解决方案

在下载Eclipse之后，其安装就比较简单了：将下载的Eclipse文件解压缩，然后就可以准备安装了。由于已经下载了符合目标操作系统的Eclipse版本，所以在解压缩Eclipse之后，可以找到相应的可执行文件。要运行Eclipse，只需运行此可执行文件即可。

讨论

Eclipse最方便的特性之一就是很容易安装。解压之后，可以就地安装。也可以进行并行安装；只需将其解压到所选的目录中，并运行可执行文件（如 *eclipse.exe*）即可。

大型的、侵入式集成开发环境（invasive IDE）的用户可以体会到安装Eclipse的便利：安装迅速，无需频繁地重启，不存在隐藏的间谍软件（spyware）。Windows开发人员可以感到宽慰，Eclipse不需要在Windows注册表中进行安装——所有干预问题都是由此引起的。所以，安装/重新安装是一件轻松的事情。

首次运行 Eclipse 时，默认情况下将显示 Resource 透视图，如图 1-2 所示。正如本章稍后的 1.6 节所述，透视图向用户呈现窗口的布局。如果反复打开某个透视图，得到的总是一组相同的窗口。Resource 透视图是一种通用的透视图，适合于资源管理，特别是文件管理。但我们一般不使用 Resource 透视图，因为我们所需的一切功能，事实上包括 Resource 透视图提供的全部功能，都包含在面向 Java 的各种透视图（特别是本章稍后讨论的 Java 透视图）中。

注意：默认透视图不一定必须是 Resource 透视图。选择 Window → Preferences → Workbench → Perspectives，可以改变默认的透视图。Java 程序员通常在这里选择 Java 透视图，这是 Java 开发的主要透视图。

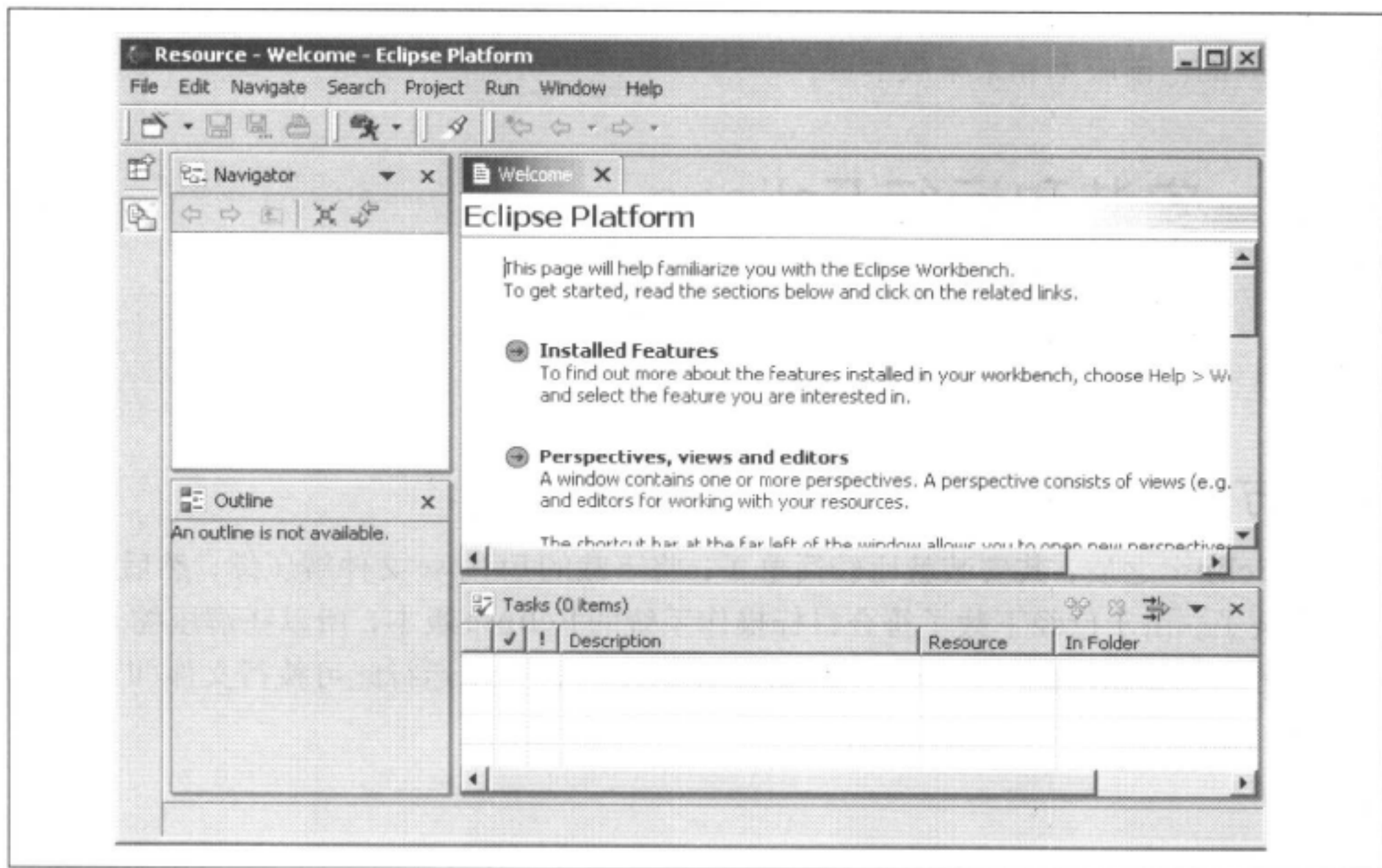


图 1-2：Resource 透视图

注意：要启动 Eclipse，计算机上必须安装有 Java。如果是首次启动 Eclipse，且出现了一个对话框，显示一条消息“A Java Runtime Environment (JRE) or Java Development Kit (JDK) must be available in order to run Eclipse”（要运行 Eclipse，必须安装有 Java Runtime Environment (JRE) 或 Java Development Kit (JDK)），请首先下载并安装 Java。可以从站点 <http://java.sun.com/j2se/> 免费获得 Java。

在安装了Eclipse之后,通过为之创建一个快捷方式,可以更方便地启动Eclipse。在Linux或Unix中,可以将Eclipse目录添加到自己的路径中,或者使用 `ln -s` 命令创建一个到Eclipse的符号连接。在Windows中,在Windows资源管理器中右击可执行文件,并从打开的快捷菜单中选择“创建快捷方式”命令,然后将新建的快捷方式拖到所需要的位置即可。

快速启动

一般情况下,Eclipse的启动相当慢,但可以改进启动性能(如果你是Eclipse新手,在获得更多的经验之前,可能需要耐心地等待)。启动速度之所以比较慢,通常是因为Eclipse有很多任务要完成。为了加速启动过程,应减少安装的插件的数量,在退出Eclipse之前应关闭视图和编辑器,并从工作区删除不必要的项目。甚至可以调整Eclipse使用的Java虚拟机(JVM)。例如,可以使用启动参数为Java虚拟机提供更多的内存,如参数 `-vmargs -Xmx512m` 可以为Java虚拟机提供512 MB的工作内存,从而消除与磁盘的大量文件交换。

参考

1.6 节,管理透视图、视图和编辑器;1.3 节,理解Eclipse工作区;Eclipse (O'Reilly) 一书的第1章。

1.3 理解 Eclipse 工作区

问题

术语“工作区”是什么意思?什么是插件?

解决方案

尽管可以将Eclipse看做是一个Java集成开发环境,但Eclipse实际上包含大量的、在Eclipse工作区的后台协同运行的组件。插件是一种软件工具,可以在Eclipse中完成特定的任务,例如,使你能够编辑Ant文件、编译Java文件或拖放GUI元素。工作区以及Eclipse工作台、团队组件、帮助组件,都是整个Eclipse平台的组成部分,是插件与Eclipse核心软件进行交互的基础。

讨论

你所使用的 Java 集成开发环境是 Java 开发工具箱 (Java Development Toolkit, JDT)。JDT 并非 Eclipse 的组成部分，而是一个插件。实际上，Eclipse 由 Eclipse 平台组成，后者为其他工具提供支持。这些工具是作为插件实现的，从而使平台本身成为一个相对较小的包。

Eclipse 带有大量的插件，包括 JDT。另一个重要的插件是插件开发环境 (Plug-in Development Environment, PDE)，使用它可以开发自定义的插件。如果需要使用 Java 以外的语言进行开发，需要使用与该语言相应的插件，这样的插件有很多。

注意：除了可以使用不同的程序设计语言以外，还可以改变 Eclipse 使用的人类语言。不同的语言通常以不同的插件段 (plug-in fragment) 来支持。对于很多语言，包括日语、朝鲜语、德语、法语、意大利语、葡萄牙语、西班牙语，甚至繁体中文和简体中文，都有可用的插件段。

了解 Eclipse 的组成，对于使用 Eclipse 的各种功能（不仅仅是最常用的功能）是十分必要的。如果对各种组件的功能一无所知，遇到问题时将不知所措，从而可能使 Eclipse 的行为变得反复无常。例如，如果你知道 JDT 不同于其他的插件，当 JDT 中可用的选项而在其他插件中没有时，就不会感到奇怪。

Eclipse 平台由下面几个组件构成：平台内核、工作区、工作台、团队组件和帮助组件。通过图 1-3 可以概括了解这些组件。插件在 Eclipse 启动时加载，所以图 1-3 中标明了一些插件。

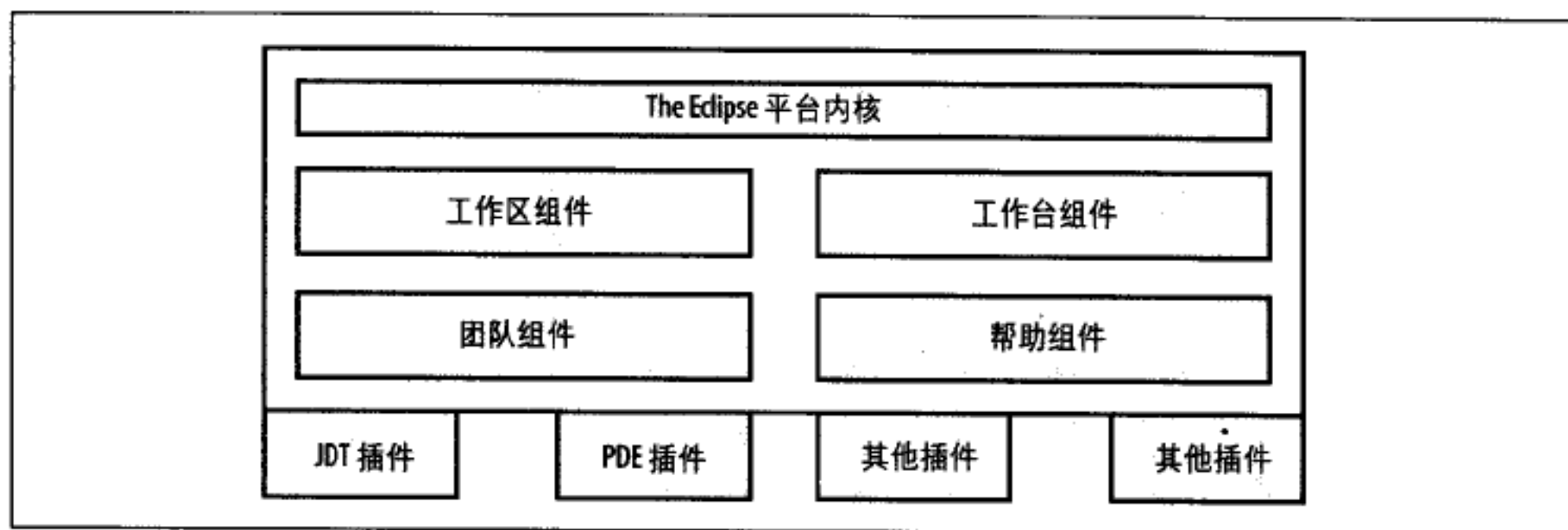


图 1-3: Eclipse 的组件和插件

Eclipse 平台内核

平台内核是所有其他功能的基础。平台内核用本机代码编写，其作用是加载平台的其余部分；如果平台内核找不到可用的Java安装，以运行平台的其余部分，内核将发出警告信息。内核还将加载 Eclipse 插件。

工作台组件

工作台基本上就是 Eclipse。工作台显示图 1-2 所示的菜单和工具栏（所显示的菜单和工具栏项目取决于当前查看的透视图）。

尽管可以加载不同的插件，而每一种插件又有不同的窗口和菜单系统，但这些都要通过工作台来显示。工作台被设计成看起来像是本机应用程序，是面向操作系统的。在Linux中，它看起来像一个Linux应用程序；在Windows，像一个Windows应用程序，等等。

注意：将 Eclipse 的图形用户界面设计成面向操作系统引起了某种程度的争议。工作台是用 Eclipse 自己的标准窗口小部件工具箱（Standard Widget Toolkit, SWT）和 JFace 包（基于 SWT 构建）构建的，SWT 和 JFace 在显示时使用操作系统本机的组件。当我们在后面的第 8 章、第 9 章和第 10 章中讨论 SWT 和 JFace 时，你会发现，在 Java 应用程序中这样做仍然是一个有争议的问题。

工作区组件

在 Eclipse 中，工作区的作用是管理资源，包括存储在磁盘上的资源。Eclipse 以项目的形式管理资源，而且在默认情况下，由工作区组件管理 Eclipse 工作区目录中的每一个项目。在本章稍后的 1.5 节将进一步介绍项目。

注意：项目可以不存储在工作区目录中，可以使用其他的目录，甚至网络上的目录。要选择不同的存储位置，在给创建的项目命名时应取消 Use default 复选框，并填写所需要使用的目录。

工作区组件管理项目中的所有资源，包括代码。它还管理对代码和其他文件的修改，使你能够访问存储的修改历史记录，甚至能够以图表的形式比较这些修改。工作区还可以与插件（如 JDT）进行通信，使你能够访问历史记录和文件信息。

团队组件

团队组件使你能够对代码进行版本控制，并支持文件共享。如果你在公司环境中开发过软件，可能已经使用过源代码控制和储存库。代码存储在一个储存库中，而且必须进行签入或签出，这意味着可以跟踪对软件的任何修改。

协调团队对代码所做的修改，就意味着可以避免随机覆盖修改，否则将导致极度混乱。Eclipse 与并行版本系统 (Concurrent Versions System, CVS) 完美地集成在一起，CVS 是一个事实上的版本控制标准（也许微软的开发工具要除外，在这里 Visual SourceSafe 仍然占据着绝对的统治地位）。事实上，团队组件可以作为 CVS 的客户端，与维护储存库的 CVS 服务器进行交互，团队成员从储存库中检索代码。我们将在第 6 章简单介绍 CVS 的工作原理。

帮助组件

帮助组件是 Eclipse 的文档系统，用于提供帮助。这是一个可扩展的帮助系统；插件可以提供自己的 XML 格式的帮助，向帮助系统说明如何在插件的文档中导航。

插件

通过插件，可以扩展 Eclipse。插件可以建立自己的透视图、编辑器、视图以及向导等等。例如，JDT 是一个插件，它将自己的功能添加到工作台已经提供的平台中。除了有 450 多种插件可以供 Eclipse 使用以外，我们还可以构建自己的插件（请参阅第 12 章）。

1.4 运行多个 Eclipse 窗口

问题

你想同时运行多个 Eclipse 窗口，可能包括不同版本的 Eclipse 窗口。

解决方案

多次启动 Eclipse，就可得到多个 Eclipse 窗口。

讨论

运行多个 Eclipse 窗口不是一个问题，如图 1-4 所示。正如下面几小节所述，有多种不同的选择。

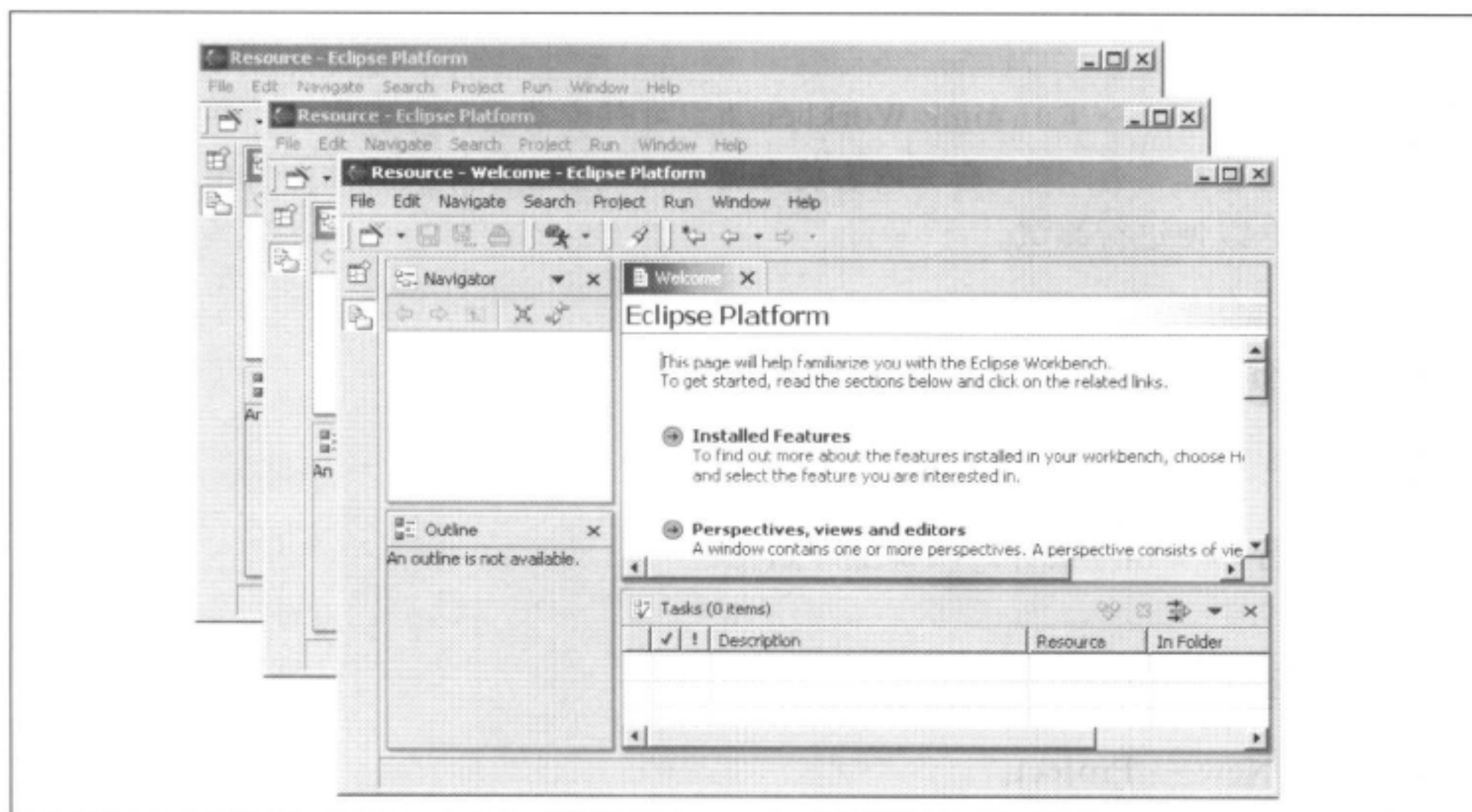


图 1-4：运行多个 Eclipse 窗口

多个 Eclipse 窗口，同一个工作区

要打开多个 Eclipse 窗口，并使用同一个工作区，可以选择 Window → New Window。如果你想在不同的窗口中同时使用两个不同的透视图（比如 Java 透视图和 Debug 透视图），用这种方法是比较合适的。

多个 Eclipse 窗口，多个工作区

可以启动 Eclipse，使其采用所选的工作区，而不是默认的工作区。为此，可以在命令行输入命令 `eclipse -data newWorkspacePath -showLocation`。此命令使 Eclipse 启动时显示由 `newWorkspacePath` 指定的工作区（`-showLocation` 选项使窗口显示其位置信息，在启动多个窗口时便于记住当前所处的窗口）。如果你想把不同的项目划分到不同的工作区中，以保持它们的独立性，应该使用这种方法。

多个 Eclipse 安装

在同一机器上可以安装多个 Eclipse，包括不同的版本。只需将它们解压到不同的目录中即可，它们不会发生冲突。如果你想试用非发布版的新特性，或者不想从命令行启动 Eclipse 以使用不同的工作区，这可能是非常有用的。

运行时工作台

选择 Run → Run As → Run-time Workbench, 可以启动运行时工作台。在试验插件时运行时工作台是非常有用的, 有关内容将在第 12 章介绍。你开发的插件将出现在这个新的工作台中, 以便进行测试。

1.5 创建一个 Java 项目

问题

你想开始进行 Java 程序设计。从哪里开始呢?

解决方案

选择 File → New → Project。

讨论

在 Eclipse 中, 所有代码 (Java 的或其他语言的) 都必须放在项目中, 而创建 Eclipse 项目是一项基本技能。项目用于组织文件、类、库和输出。在接下来的几节中, 我们将创建一个 Java 项目, 该项目将使用例 1-1 中的代码显示一些简单的文字 “Stay cool.”。

注意: 创建 Java 项目是一项基本技能, 本章的内容也全部是关于基本技能的。然而, 限于篇幅, 还有很多基本技能本章不能一一介绍。关于创建 Java 项目的详细内容, 请参阅第 3 章和第 4 章。

例 1-1: FirstApp.java 示例

```
public class FirstApp
{
    public static void main(String[] args)
    {
        System.out.println("Stay cool.");
    }
}
```

首先, 创建一个 Java 项目, 方法是在 Eclipse 中选择 File → New → Project, 然后打开 New Project 对话框, 如图 1-5 所示。

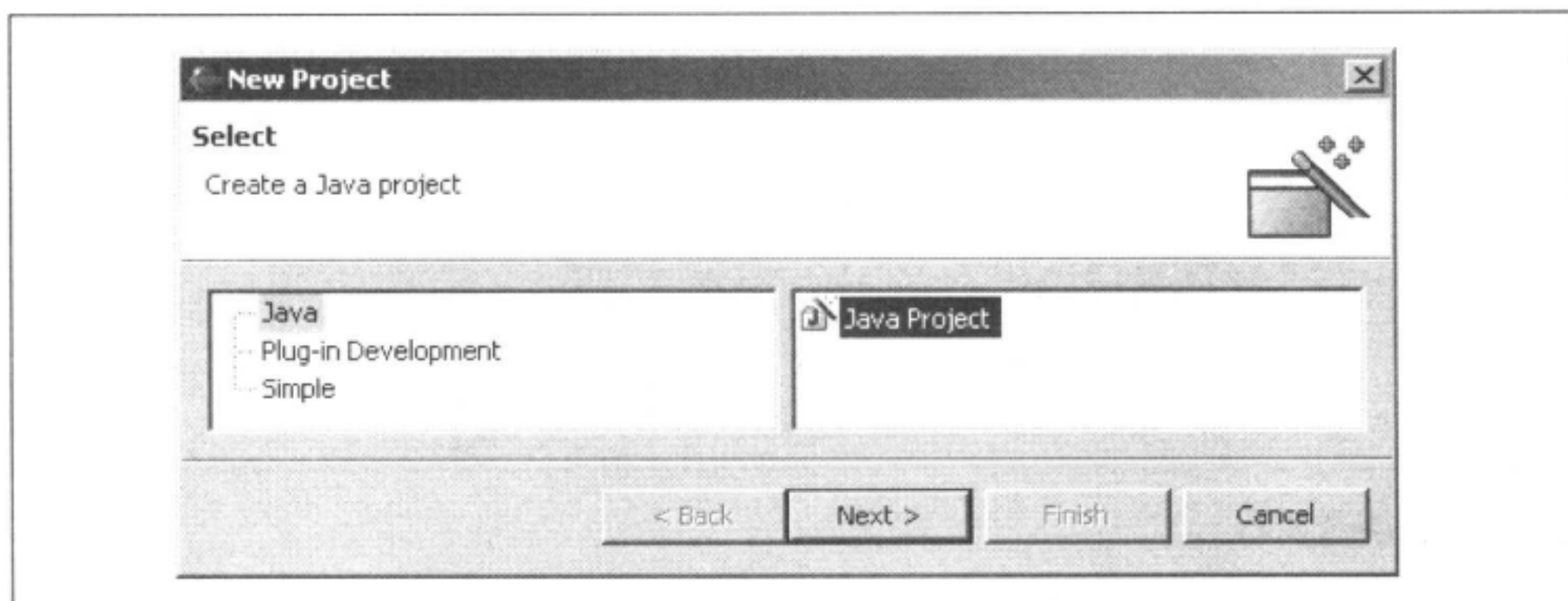


图 1-5: New Project 对话框

因为要创建的是一个 Java 项目，所以在左边的列表框中选择 Java，而在右边的列表框中选择 Java Project。单击 Next 按钮，在下一个对话框中，将项目命名为 FirstApp，如图 1-6 所示。

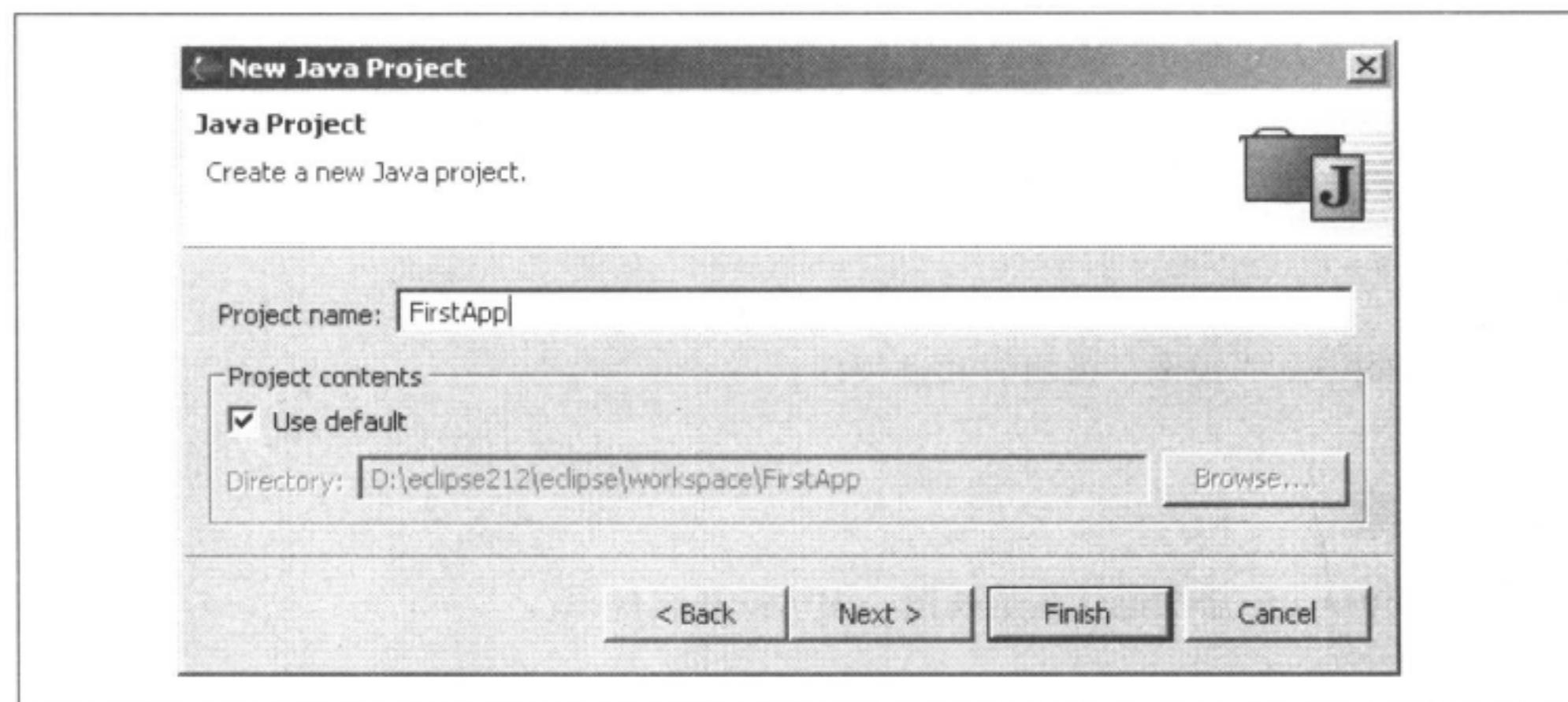


图 1-6: 为项目命名

单击 Finish 按钮，完成项目的创建（如果单击 Next 按钮，在打开的对话框中将显示创建项目的其他选项；但本章只涉及 Eclipse 的基本技能，关于项目创建选项的详细内容，请参阅第 3 章）。

如果是首次打开 Eclipse，且 / 或者 Resource 透视图是唯一打开的透视图，Eclipse 将提示你是否需要切换到 Java 透视图，如图 1-7 所示。单击 Yes 按钮。

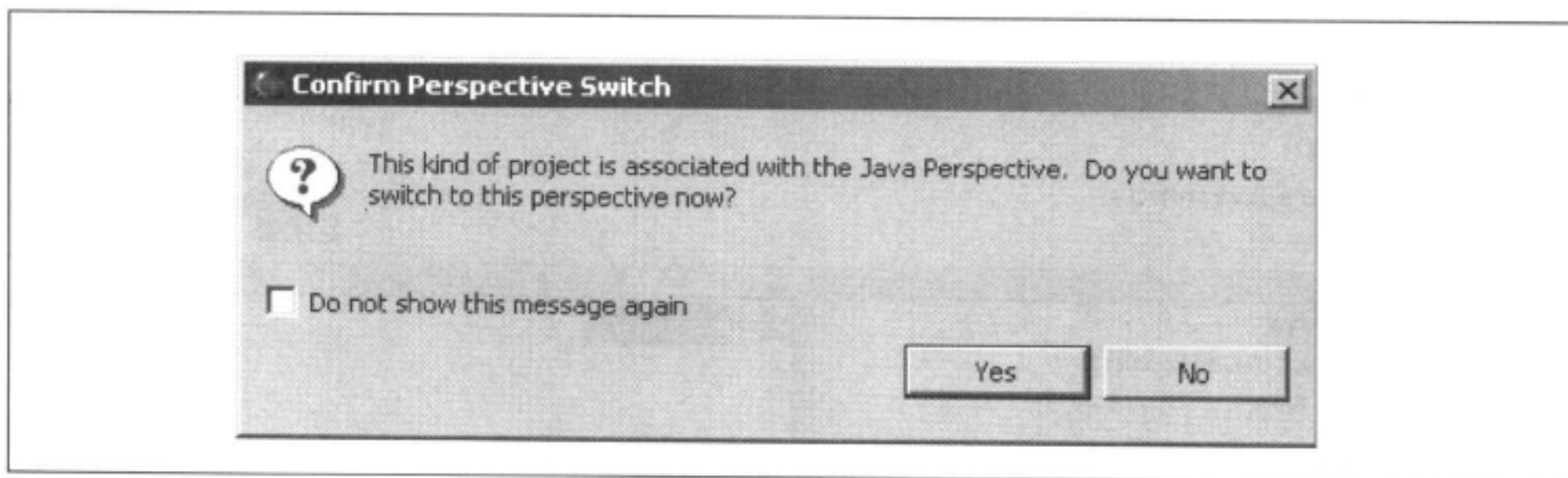


图 1-7：切换透视图

在新的项目 *FirstApp* 被创建之后，它将在 Java 透视图打开。我们已经非正式地讨论过透视图，但既然要直接使用透视图，所以在下面两节中我们将更深入地介绍一下透视图。在接下来的几节中，我们将继续开发该示例的代码。

参考

Eclipse (O'Reilly) 一书的第 1 章；1.8 节，创建一个 Java 类。

1.6 管理透视图、视图和编辑器

问题

如何使用 Eclipse 的视图、编辑器和透视图？

解决方案

要使用 Eclipse，必需知道什么是视图、编辑器和透视图：

视图

视图是一个窗口，它提供数据的图形显示，而数据可以是文本、符号列表、图形用户界面 (GUI)、图像等等。

编辑器

编辑器与视图非常相似，只是其中的数据是可编辑的。在处理代码时，在编辑器中进行编辑，编辑器通常是显示在工作台中央的那个窗口。

透视图

透视图是一组明确定义的视图和编辑器。在打开一个透视图时，它的视图和/或编辑器将出现在工作台中。

讨论

视图显示数据，但不允许编辑；编辑器既显示数据，也允许编辑数据。因为在图形用户界面中，屏幕空间始终是稀有资源，所以视图通常是叠加在一起的。通过叠加在一起的视图边缘处显示的标签，可以选择需要查看的视图。

注意： 如果需要重新打开因误操作而关闭的视图，可以选择 Window → Show View，并从出现的菜单中选择需要打开的视图。

当打开代码或其他资源时，其数据将显示在编辑器中，以便进行编辑。根据资源的文件扩展名，Eclipse 可以自动为打开的资源选择正确的编辑器：JDT 的 Java 代码编辑器适用于 Java 代码（.java 文件），XML 编辑器（如果已经安装的话）适用于 XML 文件（.xml 文件），等等。甚至可以在编辑器中打开 Microsoft Word 文档（.doc 文件）；Eclipse 通过 Windows 对象和嵌入（Windows Object Linking and Embedding, OLE）技术在编辑器中显示一个 Microsoft Word 窗口。

注意： 可以根据文件的扩展名，设置用来打开特定类型文件的 Eclipse 编辑器或外部程序。只需选择 Window → Preferences → Workbench → File Associations，选择文件类型，并将一个编辑器或程序与之关联即可（如果没有别的编辑器，可以使用 Eclipse 的默认文本编辑器）。还可以通过 Run → External Tools，将 Eclipse 外部的程序作为外部工具来运行。要添加和配置外部工具，请选择 Run → External Tools → External Tools。

在开发代码和使用其他资源时，大部分工作都是在编辑器中完成的。例如，当使用提供某种功能的插件时，通过拖放控件可以在编辑器中开发一个图形用户界面（GUI）（关于使用插件以拖放方式设计 GUI 的详细内容，请参阅第 9 章）。

可以同时打开许多编辑器，这些编辑器重叠显示在工作台中央。通过单击重叠编辑器顶部相应的标签，可以选择所需使用的编辑器（也可以选择 Window → Switch to Editor，显示可以切换到的编辑器列表）。只需单击相应标签上的 X 按钮，即可关闭编辑器（或者选择 Window → Hide Editors，在隐藏编辑器之后该命令将变为 Window → Show Editors）。

正如前面所述，透视图是一个成组打开和关闭的视图和编辑器的集合。例如，Java 透视图显示一组适合于 Java 开发的视图和编辑器。其中包括 Package Explorer 视图和 JDT 编辑器，Package Explorer 视图显示项目中 Java 文件的类层次结构，而 JDT 编辑器支持

Java 语法检查、快速修复语法错误等。另一方面，Debug 透视图显示适合于调试的视图和编辑器，包括监视窗口、输出控制台、变量监视窗口等。

透视图打开后通常单独显示（例如，当创建一个 Java 项目时，Java 透视图将自动打开），但也可以通过选择 Window → Open Perspective，然后从显示的子菜单中选择一个透视图，来显式打开 Eclipse 透视图。要关闭透视图，可以选择 Window → Close Perspective。

注意：当打开一个透视图时，其快捷工具栏上将出现一些图标（快捷工具栏位于 Eclipse 窗口的最左侧，参见图 1-2，其中出现了 Resource 视图——显示一个文件和一个文件夹——的图标）。一旦打开了一个透视图，Eclipse 将记住你至少使用了该透视图一次，并且该透视图的图标将出现在快捷工具栏上。要切换到该透视图，只需单击相应的图标即可。要删除快捷工具栏上的图标，可右击图标，然后单击 Close。通过快捷工具栏顶部的那个图标（显示一个透视图时带一个加号 +），可以打开新的透视图。

透视图已经预定义了一组内置的视图和编辑器。当选择一个透视图时，将自动显示对应的一组视图和编辑器。通过定义一组透视图，Eclipse 使开发工作便得更轻松，因为开发人员不必在每次想编写或调试代码时，去手工打开特定的透视图。开发人员还可以创建（或删除）自己的透视图。

要通过示例的方式弄清这些概念，请阅读下一节，在下一节中我们将详细剖析 Java 透视图。

参考

1.3 节，关于理解工作区。

1.7 掌握 Java 透视图

问题

什么是 Java 透视图，可以用它做什么？

解决方案

通过图 1-8，可以了解 Eclipse JDT 的 Java 透视图的组成。Java 透视图显示了项目 *FirstApp*，这是我们在前几节开发的项目（在图中左侧的 Package Explorer 视图中，可以看到该项目）。

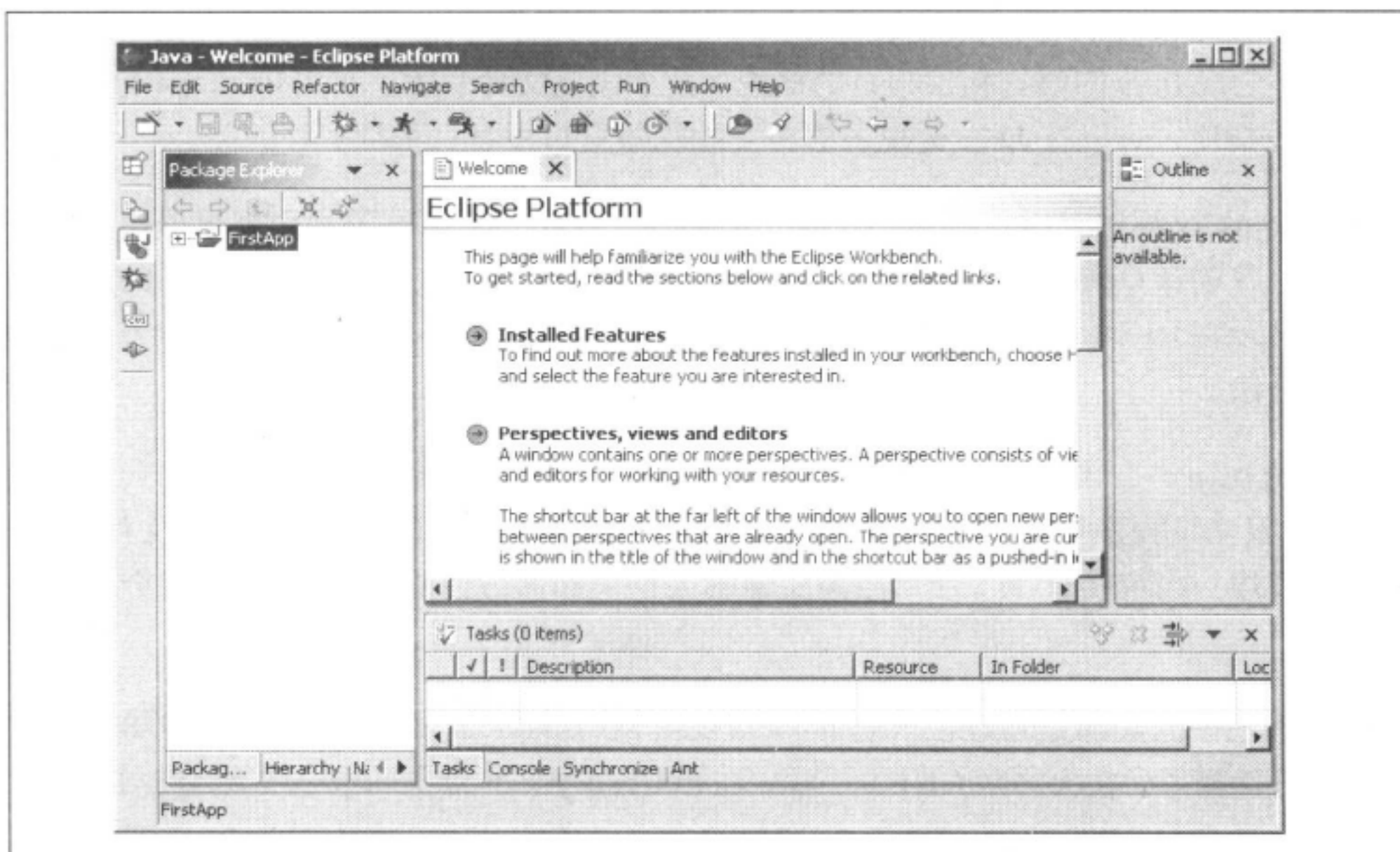


图 1-8: Java 透视图

讨论

Java 透视图是进行 Java 开发时所使用的主要的透视图；如果你打算在日常开发中使用 Eclipse，这个透视图将是你使用最多的透视图。换言之，这就是你的开发环境。

注意： 通过选择 Window → Open Perspective → Java，可以随时打开 Java 透视图。

请注意图 1-8 中的 Java 透视图。在该透视图的顶部，是标准的工作台菜单栏和工具栏，其中包含有用于 Java 开发的各种工具。在本章以及接下来的几章中，你将逐渐熟悉这些工具。

注意： 在 Eclipse 中，通过拖曳的方法可以移动工具栏。如果不想无意中移动工具栏，可以选择 Window → Lock the Toolbars，锁定工具栏。

在图 1-8 的左侧，可以看到相互重叠的 Package Explorer 视图和 Hierarchy 视图。通过视图底部的标签，可以在这两个视图之间切换。

对于 Java 程序设计而言，Package Explorer 视图尤为重要，它是 Java 开发中使用最频繁的视图；它以 Java 的语言提供项目的概貌，便于浏览文件和类。双击 Package Explorer 视图中的某项，可以在 Java 透视图的中央编辑器窗口中打开该项。

通过 Hierarchy 视图可以使用类型层次。要使用该视图，可以在透视图编辑器中右击任何一项，并选择 Open Type Hierarchy 命令。这将使该项的类型层次以可单击的继承树的形式显示在 Hierarchy 视图中。以这种方式操作项的类型层次非常适合于检查继承的方法的语法。

在 Java 透视图的右侧是 Outline 视图，其中显示编辑器中当前打开的文件中的元素的层次视图。双击该视图中的任何一项，可以跳转到编辑器中相应的项。如果代码文件很长，而且你厌烦一次滚动 20 页，使用当前文件的大纲是非常方便的。第 2 章将详细介绍这种视图。

在 Java 透视图的底部是重叠在一起的 Task 视图和 Console 视图，可以通过图 1-8 所示的标签进行选择（如在大多数 GUI 中一样，在 Eclipse 中，屏幕空间也是一种宝贵的资源，应该经常查看重叠的视图）。Tasks 视图提供未决任务的列表，作为对 Eclipse 或开发人员的提示（有一些待解决的编译错误）。Console 视图显示发送到控制台的输出，例如，我们的第一个示例应用程序将把文字“Stay cool.”输出到控制台。

在 Java 透视图的中央窗口中，显示一条 Welcome 消息，以解释透视图、视图和编辑器等概念。编辑器重叠显示在这块中央区域，可以通过顶部的标签来选择编辑器（Welcome 消息的标签如图 1-8 所示）。与视图不同的是，在编辑器中可以输入文本和数据，而 JDT 编辑器是专门为使用 Java 而设计的。除了作为输入 Java 代码的场所以外，JDT 编辑器还具有许多隐含的功能，比如语法突出显示、语法检查、自动代码填充等等。

注意：当打开 Java 透视图时，一个代表该视图的、带有小的 J 符号的图标，将出现在透视图左侧的快捷工具栏上，如图 1-8 所示。

参考

Eclipse (O'Reilly) 一书的第 1 章和第 2 章。

1.8 创建一个 Java 类 问题

你想在已有的 Java 项目中创建一个新的 Java 类。

解决方案

当打开 Java 透视图并在 Package Explorer 视图选择一个 Java 项目时，可以通过多种方式在 Eclipse 中创建新类：可以使用带有 C 图标的按钮，可以选择 File → New → Class，或者在 Package Explorer 视图中右击一个项目，并从快捷菜单中选择 New → Class。所有这些方法都将打开 New Java Class 对话框。

讨论

New Java Class 对话框如图 1-9 所示。

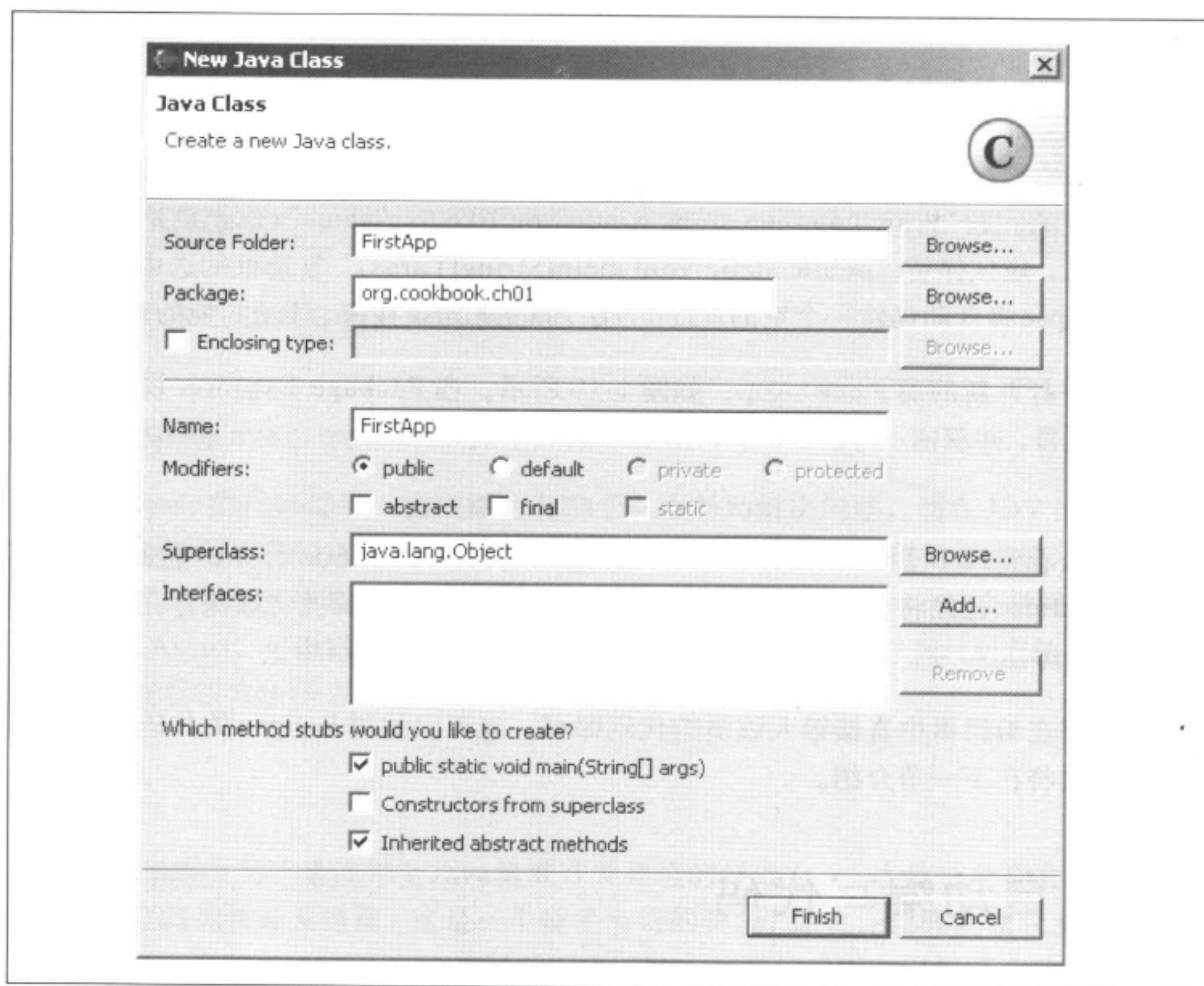


图 1-9：创建一个新的 Java 类

在前几节中，我们一直在开发一个小的 Java 项目，*FirstApp*。现在，我将使用下面的代码在该项目中显示一条消息：

```
public class FirstApp
{
    public static void main(String[] args)
    {
        System.out.println("Stay cool.");
    }
}
```

对于大多数 Eclipse 开发人员来说, 创建 Java 类是一项基本的技能, 所以本节将介绍这些基本技能。注意该对话框中的选项: 可以将类的访问说明符设置为公共的 (public)、私有的 (private) 或受保护的 (protected); 可以将类设置为抽象的或不可改变的; 可以指定新类的超类 (默认的超类为 `java.lang.Object`); 还可以指定类实现的接口 (如果有接口的话)。第 3 章将详细介绍类的创建。

在本书中, 我们将把示例放入 Java 包, 以避免与其他代码冲突; 本书以范例所在的章打头来命名包, 如 `org.cookbook.ch01`。在 Name 文本框中输入新类的名称 `FirstApp`, 而在 Package 文本框中输入该类的新包的名称 `org.cookbook.ch01`。要特别注意, 在该对话框中的问题 “Which method stubs would you like to create?” (你想创建哪些方法头?) 下方, 应该保留 “`public static void main(String[] args)`” 复选框的选中标记。这意味着 Eclipse 将自动创建一个空的 `main` 方法。单击 Finish 按钮, 接受其他的默认设置。

这将创建并打开新的类 `FirstApp`, 如图 1-10 所示。在 Package Explorer 视图中打开 `FirstApp` 项目, 并双击 `org.cookbook.ch01` 条目下的 `FirstApp.java` 条目, 以打开新类的代码。

从图中可以看到 JDT 已经编写的代码 —— 注意创建 `org.cookbook.ch01` 包的语句。还可以看到, Eclipse 已经将 `main` 方法添加到我们的类中。这个新的类将被保存在其自己的文件 `FirstApp.java` 中, 位于 Eclipse 文件夹 `workspace\FirstApp` 中。

此时, 只需在编辑器中直接输入该类的代码即可。也可以使用 Eclipse 的代码助手来简化操作, 这将在下一节介绍。

1.9 自动编写代码

问题

在输入代码时, 常常会忘记要调用的方法的名称或方法的参数。

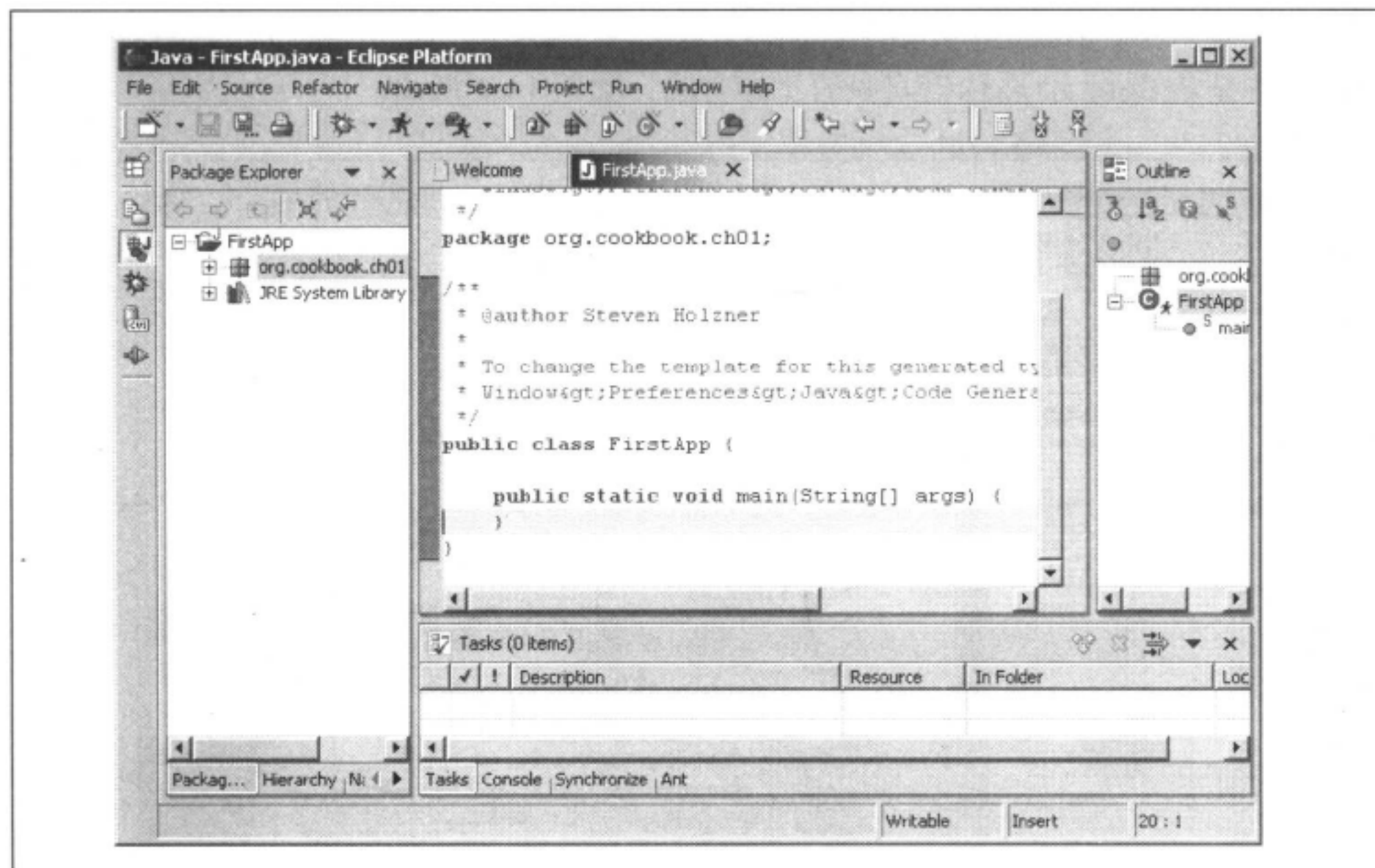


图 1-10：在 Eclipse 中打开一个 Java 类

解决方案

使用 Eclipse 的代码助手（也叫做内容助手）来帮助解决问题。当在 JDT 代码编辑器中输入一个后面带有句点（.）的对象或类的名称并暂停时，代码助手将显示该对象或类的成员，你可以从中选择所需要的那个成员。通过按 Ctrl-Space 组合键或选择 Edit → Content Assist，可以随时调出代码助手（例如，当把光标定位到一个方法的圆括号内，并希望查看该方法携带的参数时）。

讨论

代码（或内容）助手是使用全 Java 集成开发环境的好处之一。这是一个非常重要的工具，利用它可以加快开发速度；这是一个唾手可得的资源，你也许最终依靠它发现了自己的才能所在。在前面几节开发的代码示例中，输入下面的代码，以显示一些文字：

```
public class FirstApp
{
    public static void main(String[] args)
    {
        System.out.println("Stay cool.");
    }
}
```


要使用代码助手，可在 *FirstApp* 项目的 *main* 方法中输入 “System.”，然后暂停。代码助手将显示 *System* 名称空间中的类和方法，如图 1-11 所示。

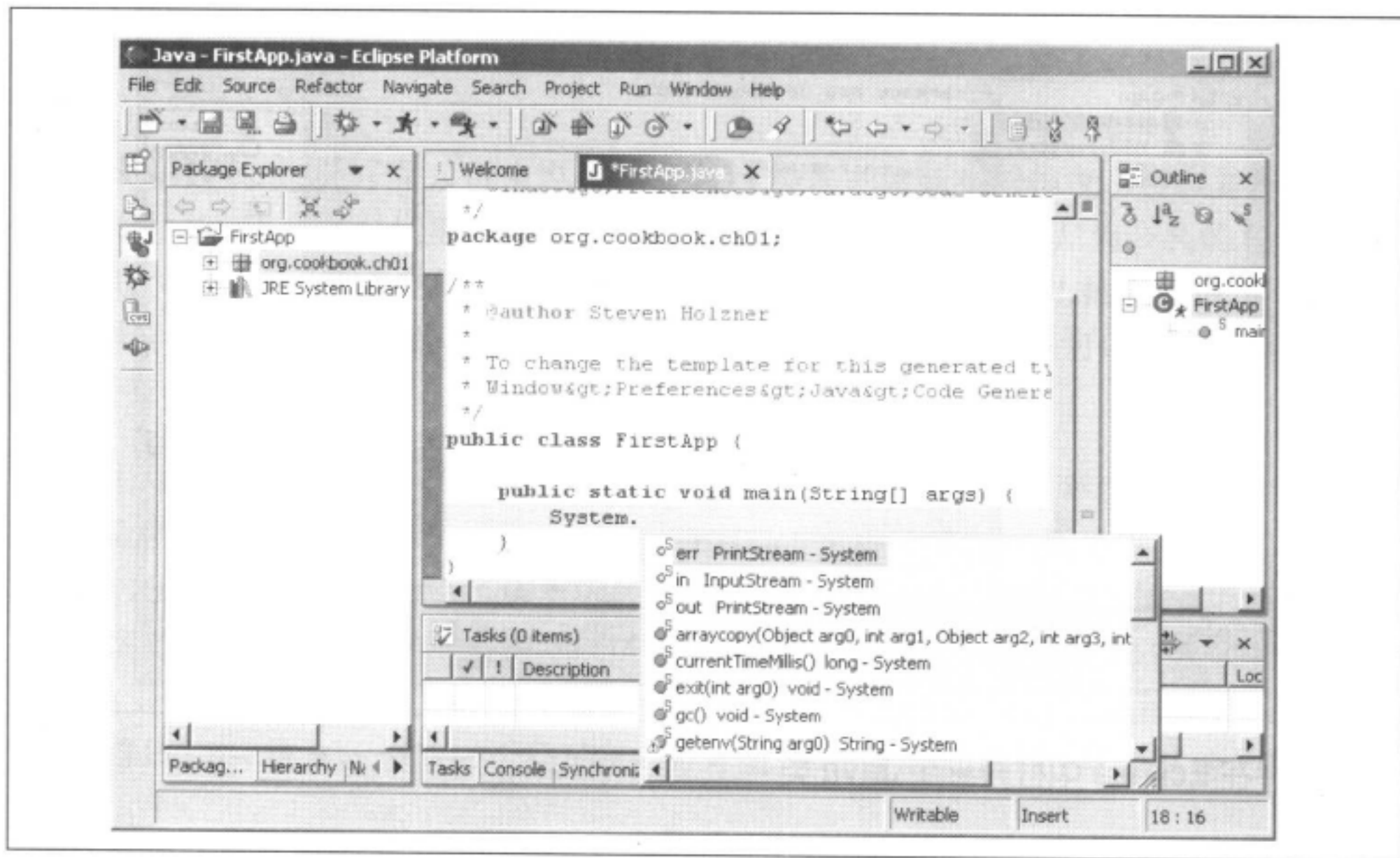


图 1-11：使用代码助手

双击代码助手列表中的 *out*，这样代码助手将把该成员插入你的代码中；插入一个句点，使插入的短语变为 “System.out.”，然后再次暂停。代码助手现在将显示 *out* 类的方法。双击代码助手给出的建议 `println(String arg0)`，代码助手把下面的代码插入到 *main* 方法中：

```
public class FirstApp
{
    public static void main(String[] args)
    {
        System.out.println()
    }
}
```

编辑上述代码，以添加文本 “Stay cool.”。注意，在输入时代码助手自动添加右引号：

```
public class FirstApp
{
    public static void main(String[] args)
    {
        System.out.println("Stay cool.")
    }
}
```

上述代码一经输入，Eclipse 即以红色波浪线显示代码，如图 1-12 所示，以表明存在语法问题。把鼠标指针停留在新输入的代码处，将显示一个工具提示，如图 1-12 所示，指示漏了一个分号。另外，在代码右侧的概述栏中出现了一个红框（图中以黑白图像显示）。单击该红框，可以跳到出错处，当代码文件很长、错误较多时这种功能是很方便的。

注意：Eclipse 不赞成使用的表达方法在 JDT 编辑器中自动加上下划线，但采用黄色而非红色。在概述栏中，语法警告通常以黄色框标示。

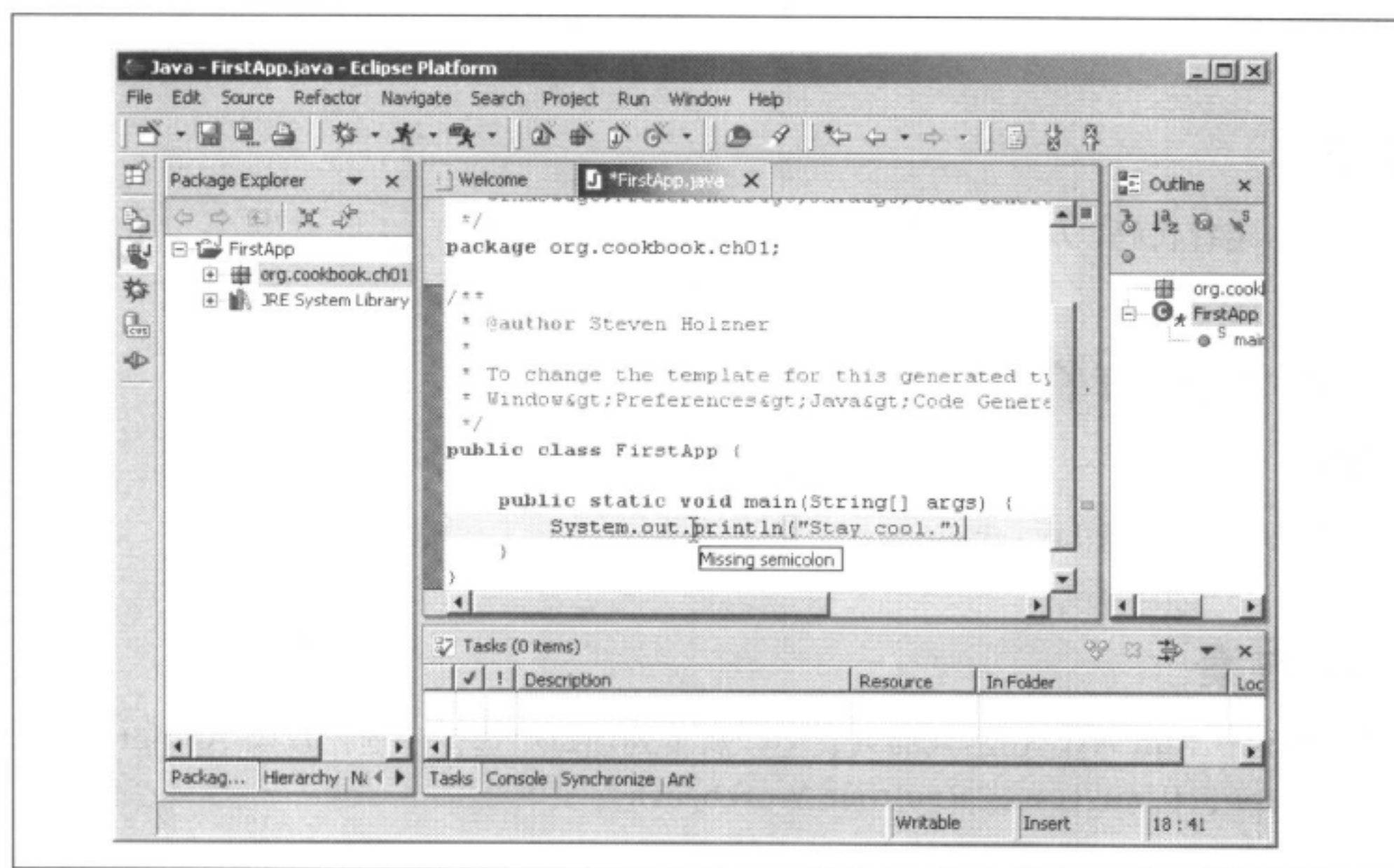


图 1-12：一条语法错误消息

现在，在该语句行的结尾处加上分号，使代码完整，并消除红色波浪线。

注意：Eclipse 可以自动格式化代码，添加缩进，并对代码进行精心的整理，在从别处粘贴代码时这种功能是非常方便的。只需选择 Source → Format 即可，Eclipse 会处理细节。最终，你会发现对这种功能的使用远远多于预期。

最后，通过单击工具栏上的磁盘图标，或选择 File → Save，保存文件。对于未保存的文件，在其编辑器标签上的文件名前有一个星号（如图 1-12 所示），文件被保存后星号

即消失。如果你未保存代码文件就试图编译和运行代码，Eclipse 将提示你保存代码文件。下一节我们将运行此代码。

总而言之，在编写代码时，代码助手是一个非常方便的工具，而且在 JDT 编辑器中在对象名或类名后加一个句点 (.) 时，代码助手将自动启动。在输入代码时，可以随时显示代码助手，只需按组合键 Ctrl-Space 或选择 Edit → Content Assist 即可。

注意：可以配置代码助手，以适合自己的编码风格。在 Preferences 对话框左侧的列表框中，选择 Window → Preferences，然后选择 Editor。然后单击 Code Assist 选项卡，即可按照自己的喜好配置代码助手。

参考

1.10 节，运行代码；*Eclipse* (O'Reilly) 一书的第 1 章。

1.10 运行代码

问题

如何从 Eclipse 运行 Java 代码？

解决方案

选择 Run → Run As。在出现的列表中，从下列项目中选择一项：Java Applet、Java Application、JUnit Test 或 Run-time Workbench。

讨论

要运行前几节开发的代码，可选择 Run → Run As → Java Application（如果尚未保存文件的话，Eclipse 将提示保存文件）。图 1-13 显示了运行结果，代码的输出 “Stay cool.” 出现在底部的 Console 视图中。

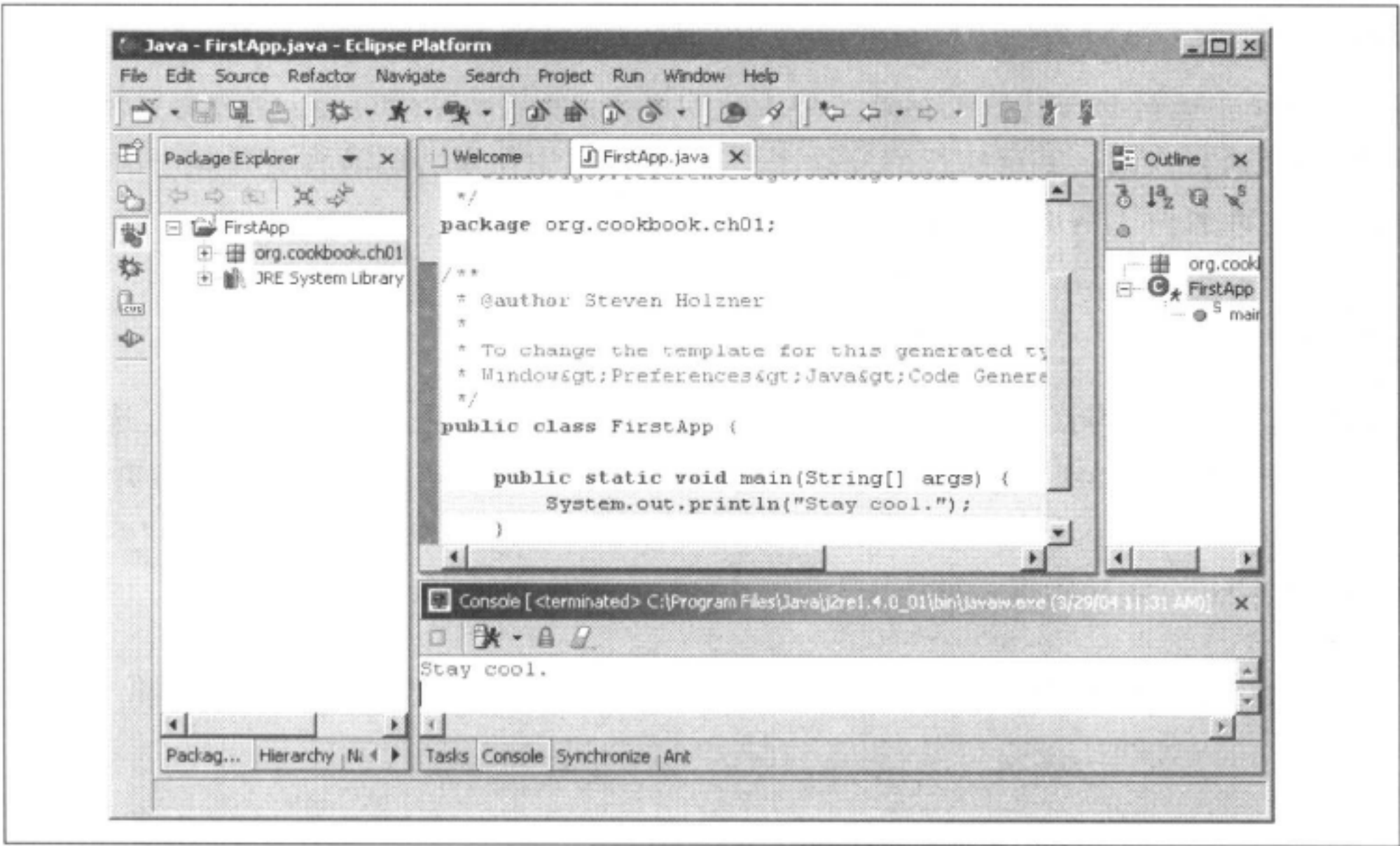


图 1-13: 运行 Java 代码

注意： 在首次运行一个 Java 程序之前，应该检查一下 Eclipse 被设置成使用是 JRE，还是 JDK。默认情况下，Eclipse 将找到已安装的 JRE 或 JDK，但它可能不是你想使用的开发环境（例如，Eclipse 可能已经找到了为你所使用的浏览器安装的 JRE）。要查看 Eclipse 使用的开发环境是 JRE 还是 JDK，可选择 Window → Preferences，打开工作台的首选参数。在左侧的树形结构窗格中，选择 Java → Installed JRE，以显示 Installed Java Runtime Environments 参数页。Eclipse 使用的 JRE 或 JDK 已经被选中；你可以选择其他的 JRE 或 JDK，供 Eclipse 使用。在下一章，我们将讨论如何添加其他的 JRE 或 JDK。

至此，你已经开发并运行了你的第一个 Java 示例。

1.11 运行代码片段

问题

你只想测试一段代码，不想运行整个程序。

解决方案

在 Eclipse 中运行 Java 代码的另外一种方便的方式：使用剪贴簿页面。通过剪贴簿页面，可以快速执行代码，甚至执行部分程序。在 Eclipse 中使用剪贴簿页面并非一项基本技能，但对此有所了解还是有益的。

讨论

在创建了剪贴簿页面之后，可以将代码复制并粘贴到该页面上，然后运行代码。选择 File → New → Scrapbook Page 即可创建剪贴簿页面。在 File 文本框中输入剪贴簿页面的名称，如 ScrapPage，然后单击 Finish。新的剪贴簿页面作为 *ScrapPage.jpge* 存储在 Package Explorer 视图中，如图 1-14 所示。

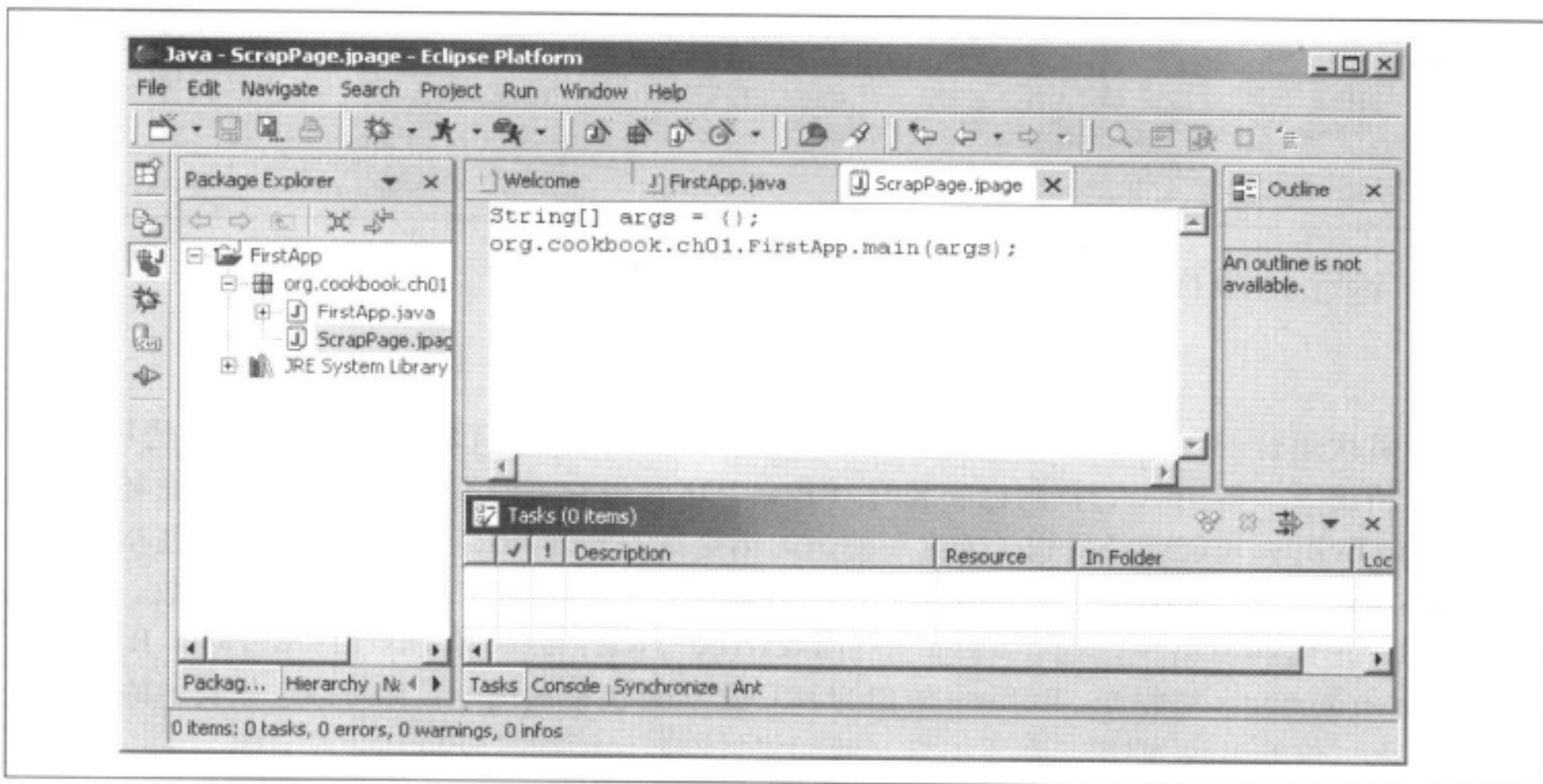


图 1-14：使用剪贴簿页面

在这个新的页面上输入要运行的代码。例如，要运行刚才运行过的 *FirstApp* 项目中的示例代码，可以在剪贴簿页面上输入下面的代码；注意，要到达示例中的 `main` 方法，必须用包的名称来限定方法的名称：

```
String[] args = {};  
org.cookbook.ch01.FirstApp.main(args);
```

通过加亮显示，可以选择要在剪贴簿页面中运行的代码。在本例中，全选剪贴簿页面中的代码，右击代码，然后单击 `Execute`，或选择 `Run` → `Execute`（如果需要导入剪贴簿页面上的代码，可以右击剪贴簿页面上的代码，然后从快捷菜单中选择 `Set Imports`）。

注意：Execute 命令可以运行剪贴簿页面上的代码，但还有其他的选择。如果选择 Display，所选代码的返回值将出现在剪贴簿页面上。当不需要用 println 语句点缀代码时，这种方法是非常有用的。

1.12 自动修复语法错误

问题

Eclipse 可以指出代码中的语法错误，但你不必非要到 Java 文档中去查找正确语法。

解决方案

Eclipse 的 Quick Fix 可以提供解决方案。当在 JDT 编辑器左侧的标记栏上发现错误 / 灯泡图标时，单击此图标可获得针对该错误的解决方案，可以从中选择所需的解决方案。

讨论

Quick Fix 是一个非常有用的工具；就我所关心的而言，即使 Eclipse 没有其他的功能，仅仅凭借其 Quick Fix 功能，它仍然是一个有使用价值的程序。在 JDT 的所有内置组件中，Quick Fix 是程序员真正喜爱的工具，因为它使程序员几乎立刻就能修复语法错误。

例如，假设我们想让前几节开发的示例代码也能显示日期和时间，可以使用下面的代码：

```
public class FirstApp {  
    public static void main(String[] args) {  
        Date date = new Date();  
        stayCoolText = "Stay cool.";  
        System.out.println(stayCoolText + " Date now is " + date.toString());  
    }  
}
```

你可能发现其中有几处错误；Eclipse 肯定也发现了错误，正如图 1-15 中编辑器左侧的标记栏上的 X 图标所示。

尽管 Eclipse 指出了存在的错误，但它并不会让你感到无助。如果 Eclipse 有一个 Quick Fix 解决方案，一个灯泡图标将出现在 X 图标旁边，如图 1-15 所示。

要处理第一个错误，可将鼠标指针在标记栏上的灯泡图标上方停留片刻，会弹出一个工具提示，指出无法解析 Date 类，如图 1-16 所示。

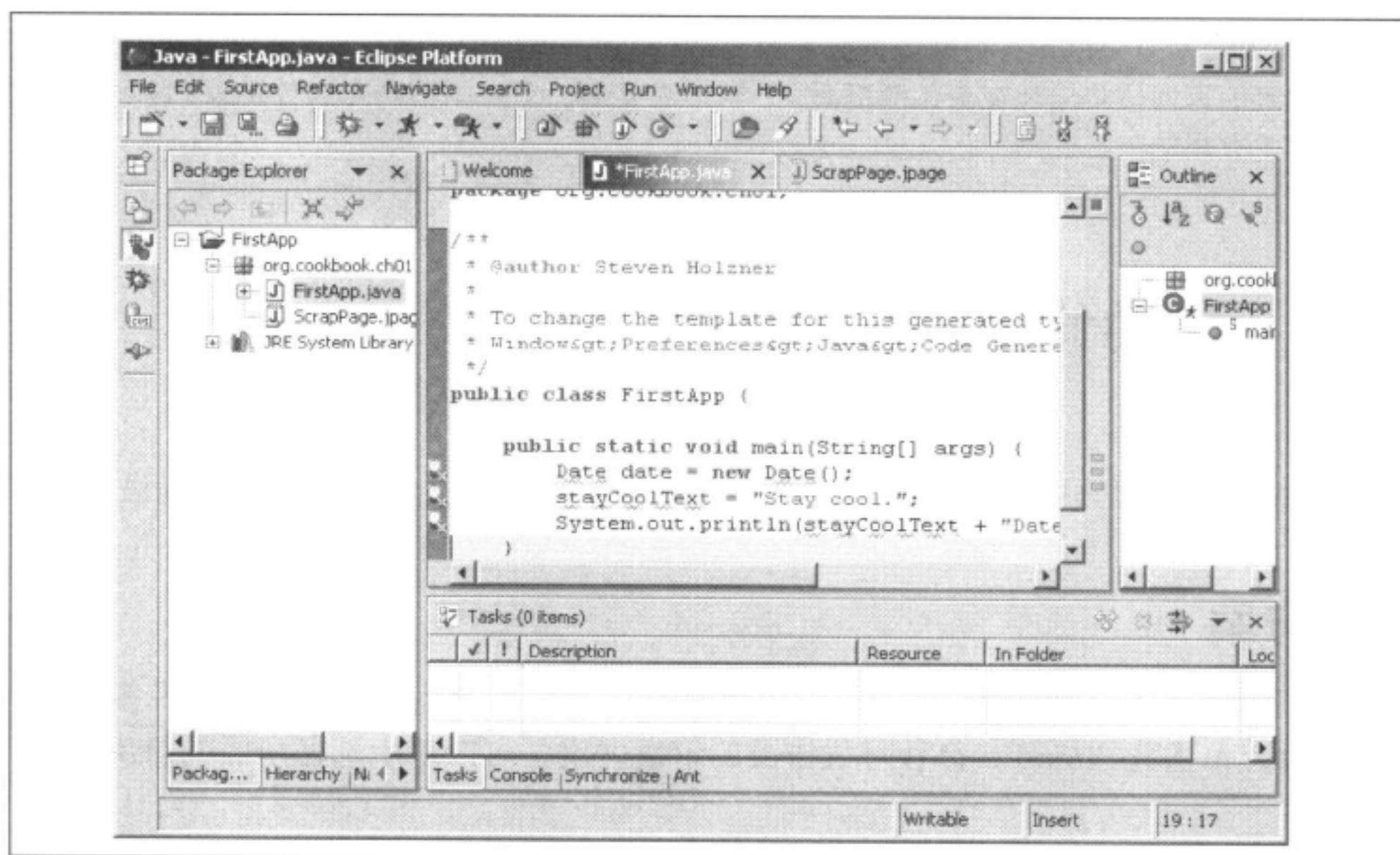


图 1-15: 使用 Quick Fix

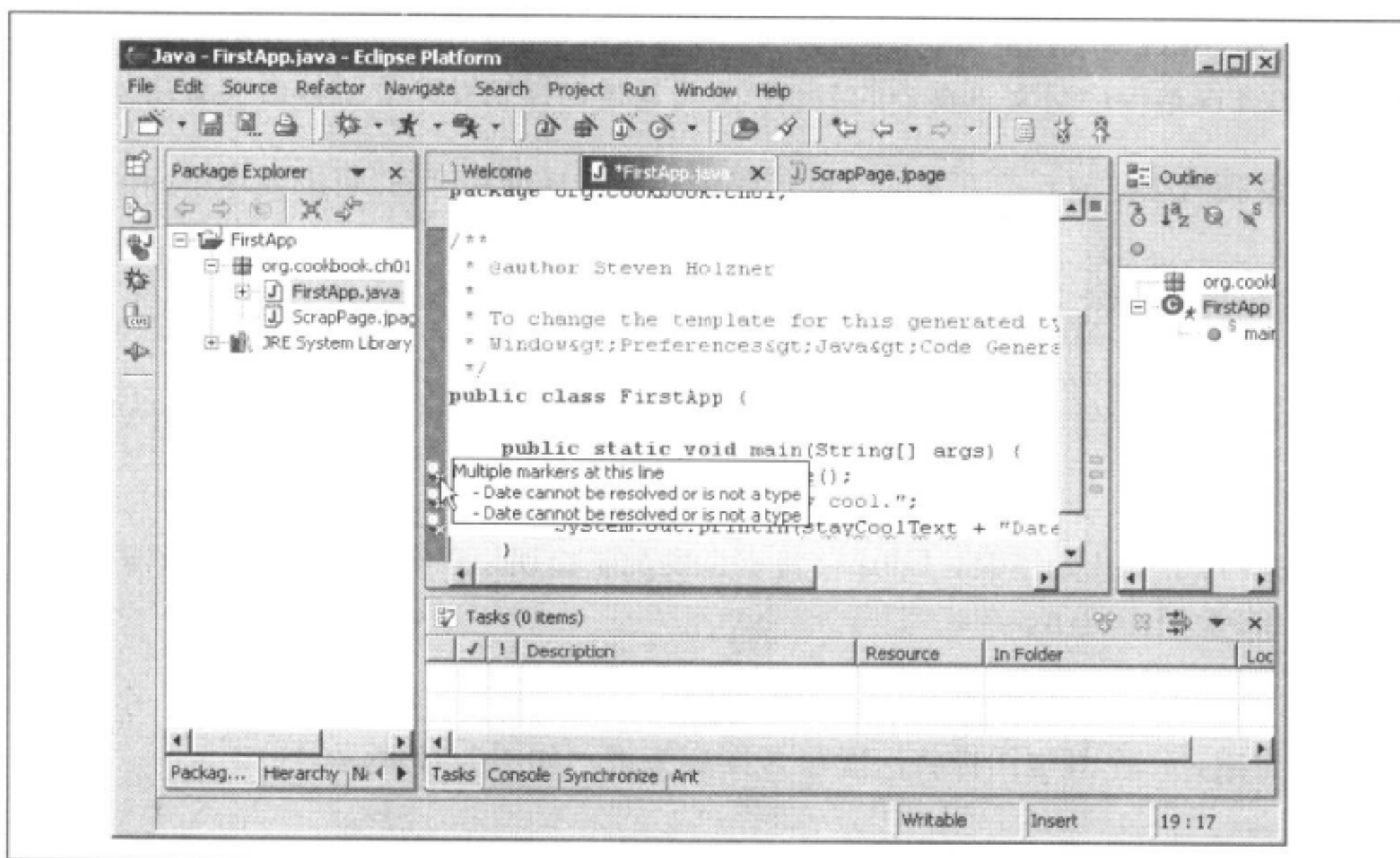


图 1-16: 获取关于错误的描述

要启动 Quick Fix，并查看 JDT 提供的建议，可单击灯泡图标。在图 1-17 中，可以看到这些建议；注意，Quick Fix 还给出了其建议添加的代码。双击建议 Import 'java.util.Date'，以导入 Date 类，使之能够被解析。

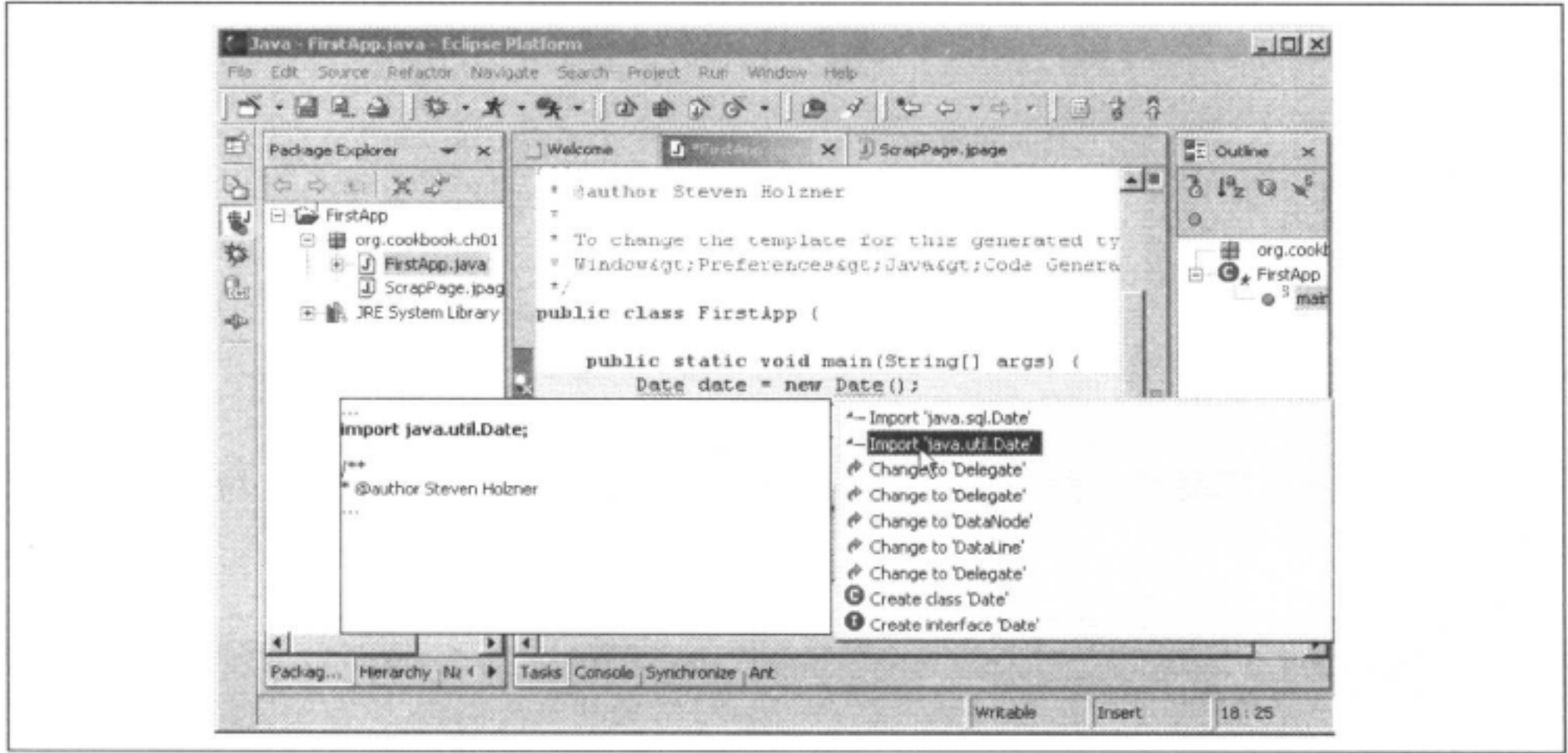


图 1-17：Quick Fix 建议

这样就解决了代码中的第一个问题。下一个问题是在使用变量 stayCoolText 之前没有先进行定义。在图 1-18 中可以看到 Quick Fix 给出的建议；双击 Create local variable 'stayCoolText' 选项。

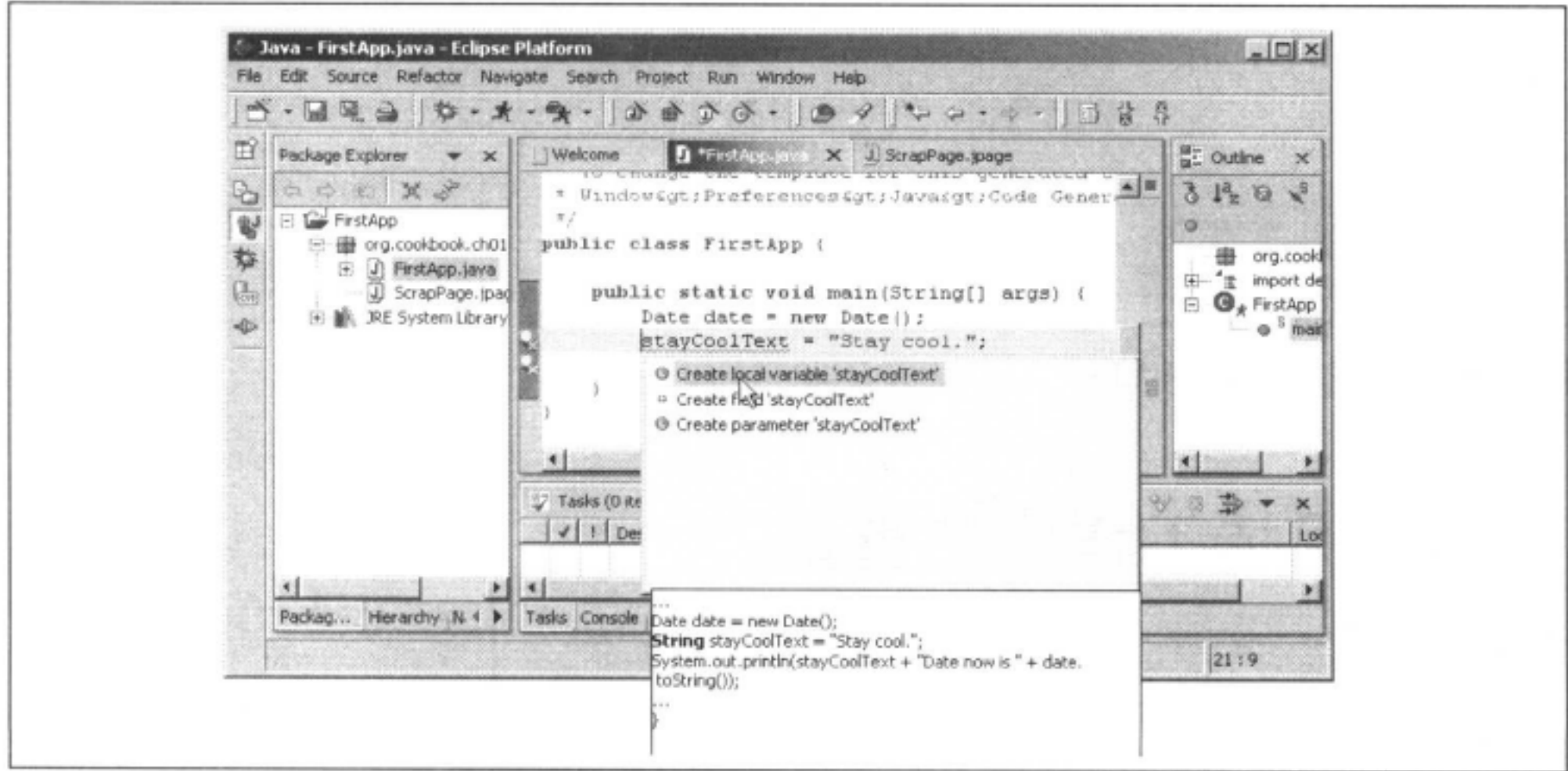


图 1-18：修复错误

存在第三个错误是因为还没有声明变量 `stayCoolText`，现在这个问题已得到解决。下面是最终修复的代码：

```
import java.util.Date;

.
.
.
public class FirstApp {

    public static void main(String[] args) {
        Date date = new Date();
        String stayCoolText = "Stay cool.";
        System.out.println(stayCoolText + " Date now is " + date.toString());
    }
}
```

在使用 Quick Fix 修复了这些问题之后，再次运行代码，如图 1-19 所示，现在没有任何问题了。

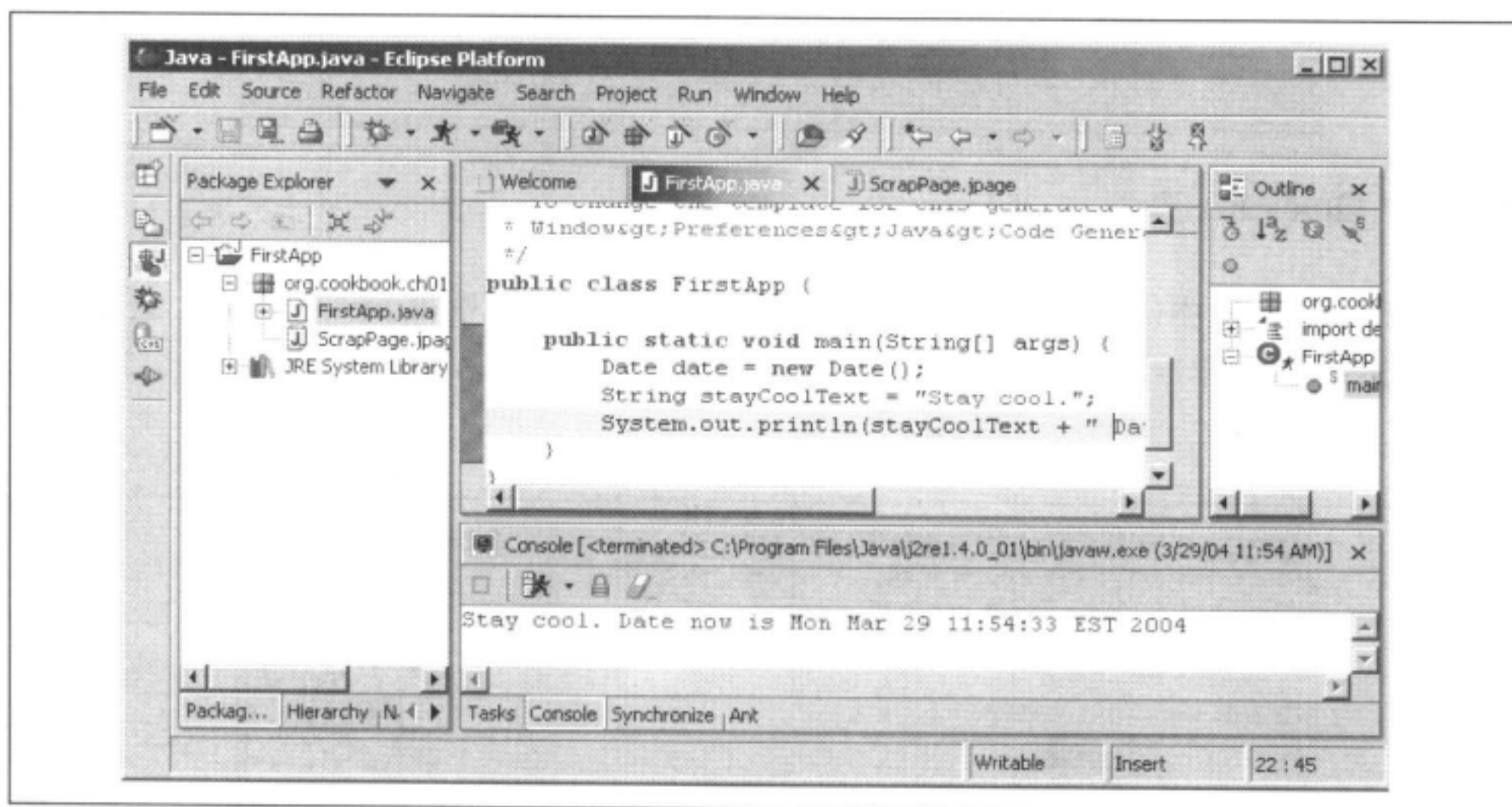


图 1-19：运行修复后的代码

1.13 保持工作区的整洁

问题

在创建了多个项目之后，工作区变得杂乱无章。结果，Package Explorer 视图包含了许多项目，包括旧的和新的，必须滚动视图才能找到所需要的项目。

解决方案

要清除 Package Explorer 中的项目，删除它即可。

讨论

删除项目时可以不删除项目使用的实际文件。需要时，可以再将项目添加回来。例如，要删除 *FirstApp* 项目，只需右击它，然后单击 Delete 即可。Eclipse 将显示 Confirm Project Delete 对话框，如图 1-20 所示。

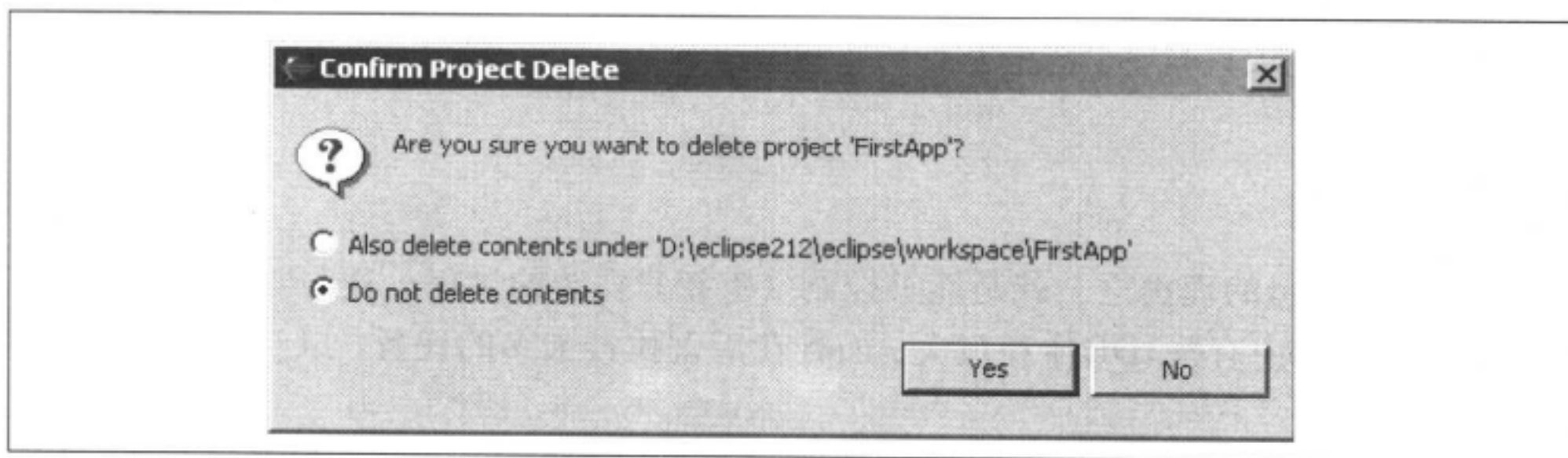


图 1-20：删除项目

确保选中 Do not delete contents 单选按钮，然后单击 Yes，从 Eclipse 中删除项目。该项目将从 Package Explorer 视图中消失。

注意：单击该对话框中的另一个单选按钮，将使 Eclipse 删除项目中的所有文件及其内容，所以，如果以后还打算使用项目的话，请不要选择该单选按钮。

如果需要再次使用项目，只需将其导入工作区即可。右击 Package Explorer 视图，从快捷菜单中选择 Import，或者选择 File → Import，可以打开 Import 对话框。选择 Existing Project into Workspace，然后单击 Next。在下一个对话框中，单击 Browse 按钮，选择 *FirstApp* 文件夹，并单击 OK。

单击 Finish，再次导入 *FirstApp* 项目。这个项目重新出现在工作区中。以这种方式从工作区中删除项目，并在需要时重新导入删除的项目，是消除工作区杂乱无章状况的最容易的方式。在第 3 章中讨论工作集时，将进一步讨论这个问题。

参考

3.17 节，创建工作集。

1.14 从灾难中恢复问题

你的 Eclipse 安装彻底损坏。

解决方案

只需保存工作区目录中需要保留的所有项目，删除解压缩文件和目录，然后重新解压下载的 Eclipse 文件即可。在替换了保存在工作区目录中的项目之后（首先忽略可能引起问题的任何可疑项目），问题就解决了。

讨论

人们喜爱 Eclipse 的理由之一就是能够控制（重新）安装的过程。这与其他 IDE 形成了鲜明的对比，其中有些 IDE 体积庞大，而且在后台执行太多的任务，以至于无法信任它们。

注意，还可以将项目从 Eclipse 的一个版本迁移到另一个版本中，为此，只需将项目文件夹复制到新版本的工作区目录中即可。对于 Eclipse 的重大修订版，迁移并非总能实现；但对于次要修订版，迁移没有任何问题。

注意： Eclipse 本身的安装不需要借助于“Windows 安装程序”，所以，无须使用 Windows “控制面板”的“添加/删除程序”项来管理 Eclipse 的安装。

使用 Eclipse

2.0 简介

本章介绍如何在日常应用中掌握 Eclipse。第1章介绍了一些基础知识，本章将介绍一些应用知识。

与任何复杂的工具一样，随着用得越来越多，你可能想改变 Eclipse 的设置。所以，除了介绍如何使用 Eclipse 以外，本章还将介绍自定义 Eclipse 的各种方式，从移动视图到创建自定义透视图。

当自定义 Eclipse 时，有一个对话框，即 Preferences 对话框，在其他对话框之上突出显示，通过选择 Window → Preferences，可以打开此对话框。该对话框如图 2-1 所示。

建议你熟悉这个对话框。Preferences 是自定义 Eclipse，特别是自定义工作台的起点。如果想在退出 Eclipse 时关闭所有编辑器（使 Eclipse 下次更快地启动），可选择 Window → Preferences → Workbench → Editors，然后选中 Close all editors on exit 复选框。如果想使编辑器标签出现在编辑器出口的底部，可选择 Window → Preferences → Workbench → Appearance，然后单击 Bottom in the Editor 制表位置框。如果想指定 Eclipse 打开某种类型的文件的编辑器或程序，可以选择 Window → Preferences → Workbench → File Associations，然后选择文件扩展名，并单击 Add 按钮即可。问题解决了！

但由于 Preferences 对话框的功能十分强大，以上这些只是自定义的初步。自定义 Eclipse 的方式有数千种。请继续阅读，以便掌握根据需要自定义 Eclipse 的方法。

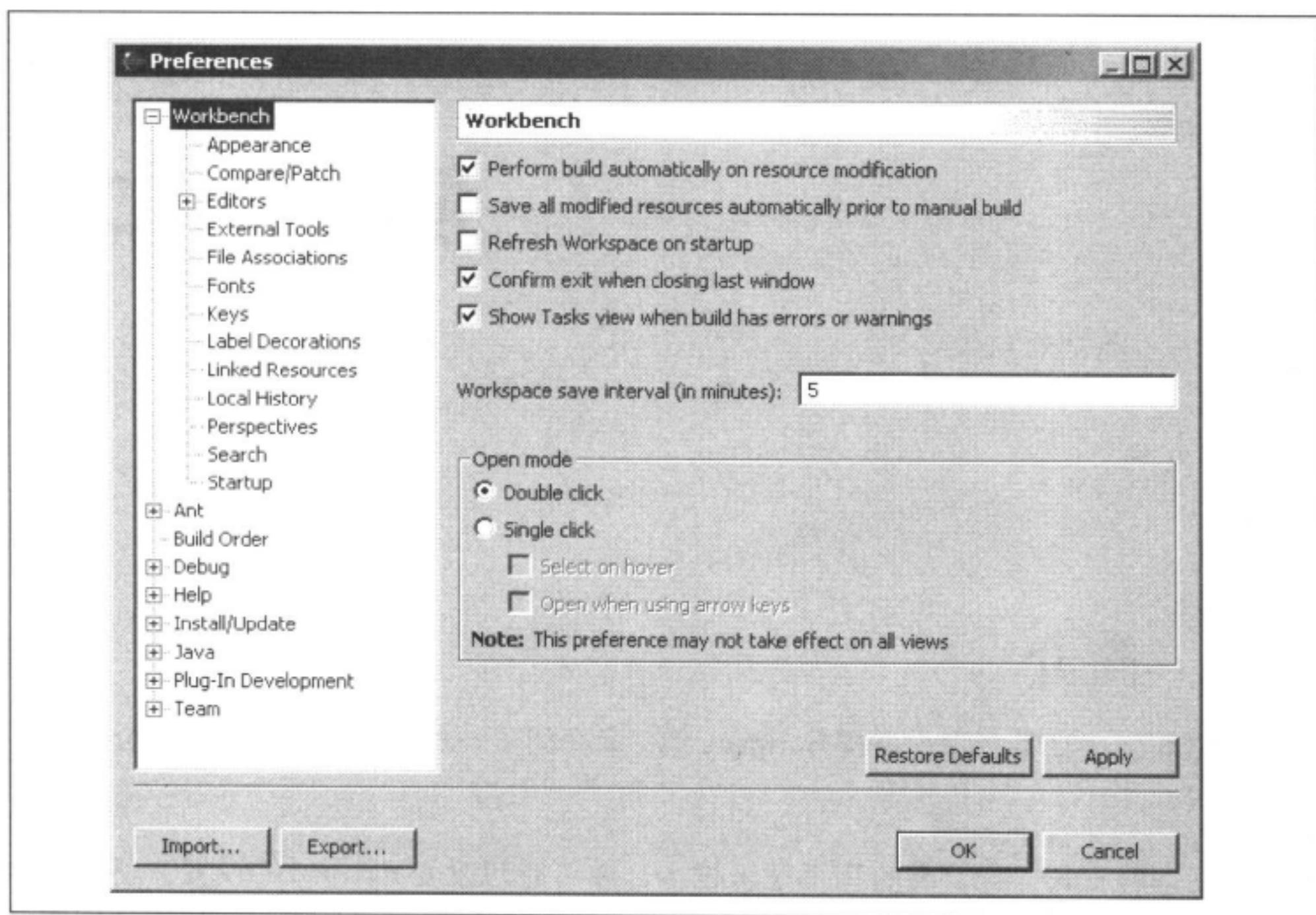


图 2-1: Preferences 对话框

2.1 显示或隐藏视图

问题

Console 视图跑哪里去了？刚才还在这里。

解决方案

要显示一个视图，可以选择 Window → Show View，然后选择需要显示的视图。如果所需要的视图不可见，可选择 Window → Show View → Other，然后从 Eclipse 显示它所知道的所有视图的对话框中，选择一个视图。要关闭视图，可单击其标签上的 X 按钮。

讨论

意外地关闭了一个视图，或者在使用透视图时其中没有显示常用的视图（如 Console 视图，该视图显示发送到输出控制台的文本），都是很有可能的。只需选择 Window → Show View，即可打开该视图。

例如，为了将 Navigator 视图添加到 Debug 透视图，可以通过选择 Window → Open Perspective → Debug，打开 Debug 视图（将在第 5 章中介绍）。然后通过选择 Window → Show View → Other → Basic → Navigator，添加 Navigator 视图。

注意： 如果想保存重新配置的透视图，可以选择 Window → Save Perspective As。详细内容请参阅本章后面的 2.21 节。

有些透视图可能会堆叠太多的视图，从而使滚动到正确的视图标签变得十分不便。既然你已经知道了在需要时可以重新打开视图，就应该毫不犹豫地关闭多余的视图，以保持屏幕整洁。

参考

5.3 节，启动调试会话；*Eclipse* (O'Reilly) 一书的第 1 章。

2.2 移动视图或工具栏

问题

Package Explorer 视图应该在右边，不是吗？

解决方案

在 Eclipse 中，可以拖动工具栏和视图，而工具栏和视图可以自动停靠在 Eclipse 窗口的各个边缘处。图 2-2 显示了被拖动到新位置的 Package Explorer 视图。

Package Explorer 视图现在出现在右侧，如图 2-3 所示。

除了拖动以外，还可以将工具栏拆分成几段。每个工具栏都有一个手柄（工具栏左边缘处的三维竖条）。通过拖动这个手柄，可以调整工具栏各段的长度，以显示或隐藏其他控件。

注意： 如果你不喜欢在工具栏会意外移动的环境中从事开发工作，那么可以选择 Window → Lock the Toolbars，以锁定工具栏。如果透视图因鼠标的偶然移动而变得杂乱，可以选择 Window → Reset Perspective，以还原透视图。

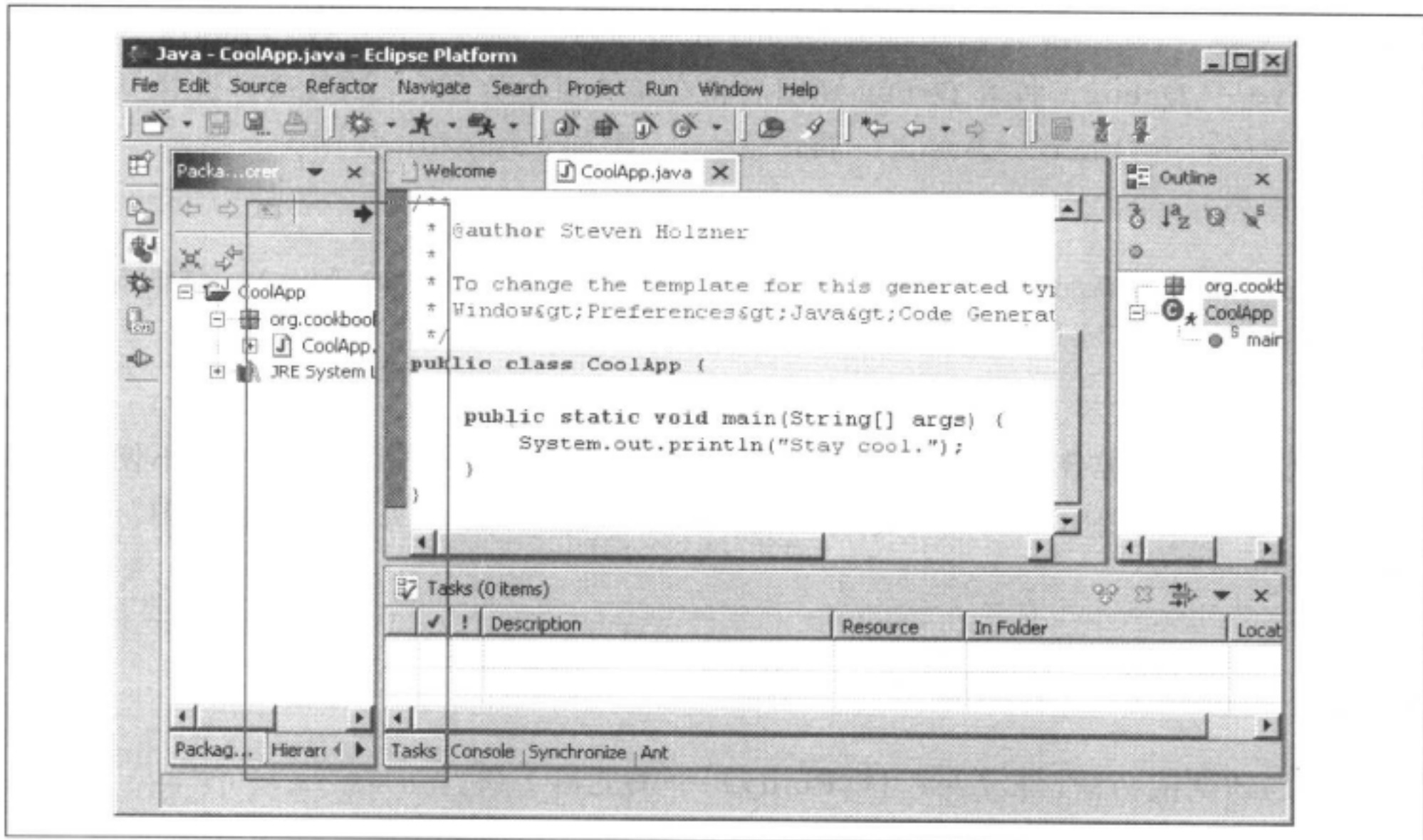


图 2-2: 拖动一个视图

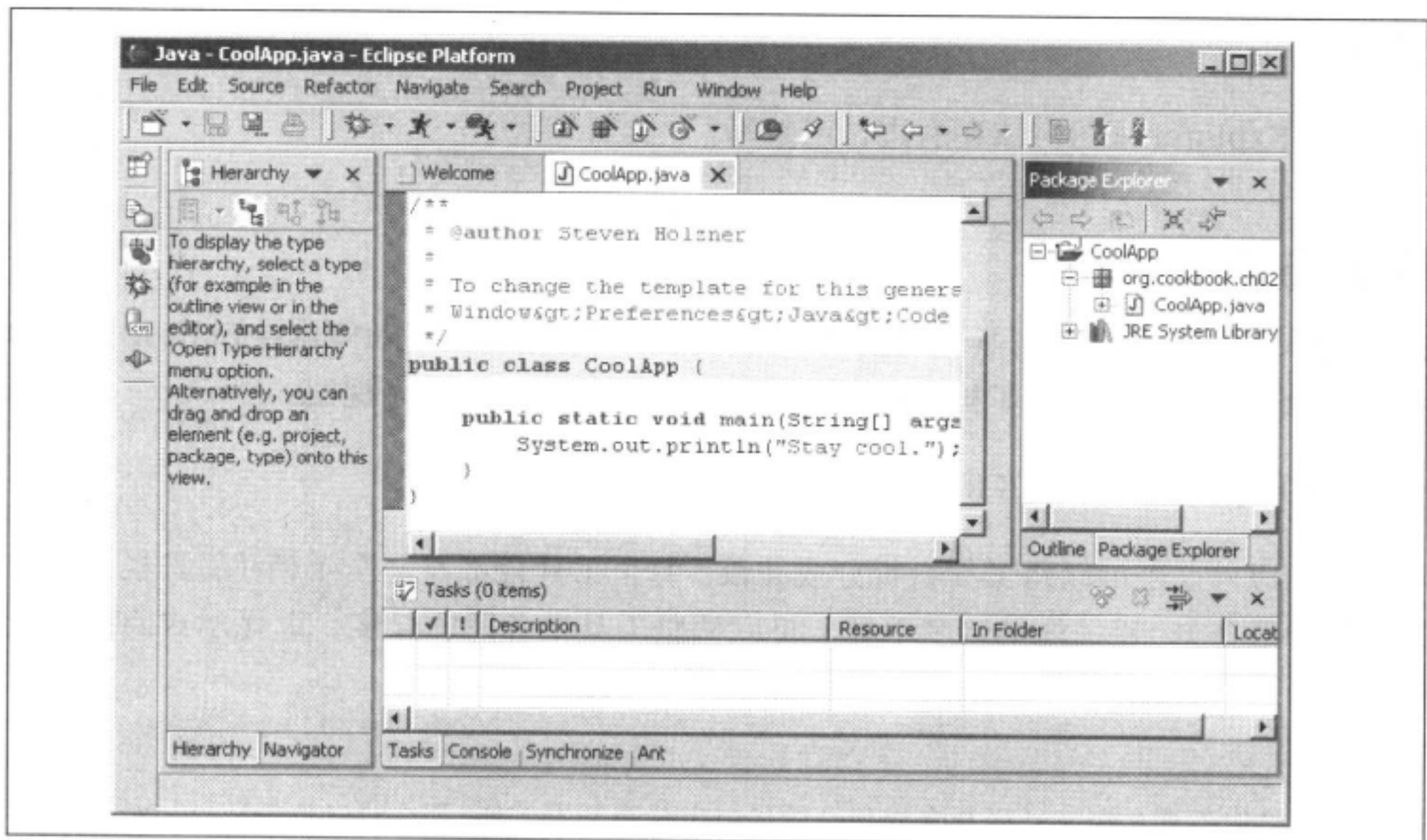


图 2-3: 重新定位 Package Explorer 视图

讨论

编辑器也可以拖动，但不能混合编辑器和视图区域中的内容。编辑器区域位于Eclipse窗口的中央，而且Eclipse不允许将任何视图拖到这里，也不允许把编辑器拖动到一个视图的上方。

参考

Eclipse (O'Reilly) 一书的第1章。

2.3 访问任何项目文件

问题

许多透视图都把文件隐藏起来，而你想访问项目中的某个（某些）文件。

解决方案

通过Resource透视图中的Navigator视图，可以访问项目中的所有文件，不存在例外。

讨论

有些透视图会隐藏文件。例如，Eclipse把项目信息存储在一个名为`.project`的XML文件中，但许多视图，如Java透视图的Package Explorer视图，把该文件隐藏了起来。在Resource透视图打开的示例项目的`.project`文件如图2-4所示。

有些面向Java的透视图也支持Navigator视图。许多Java程序员完全忽略了Resource透视图，但一些面向Java的视图可以隐藏各种文件。要访问所有这些文件，不应忘记使用Navigator视图，在Resource透视图以及一些Java透视图，Navigator视图始终是可用的。

2.4 平铺编辑器

问题

尽管通过单击编辑器标签可以切换编辑器，但有时更希望能同时打开两个编辑器，例如，当需要直观地比较两个文件时。

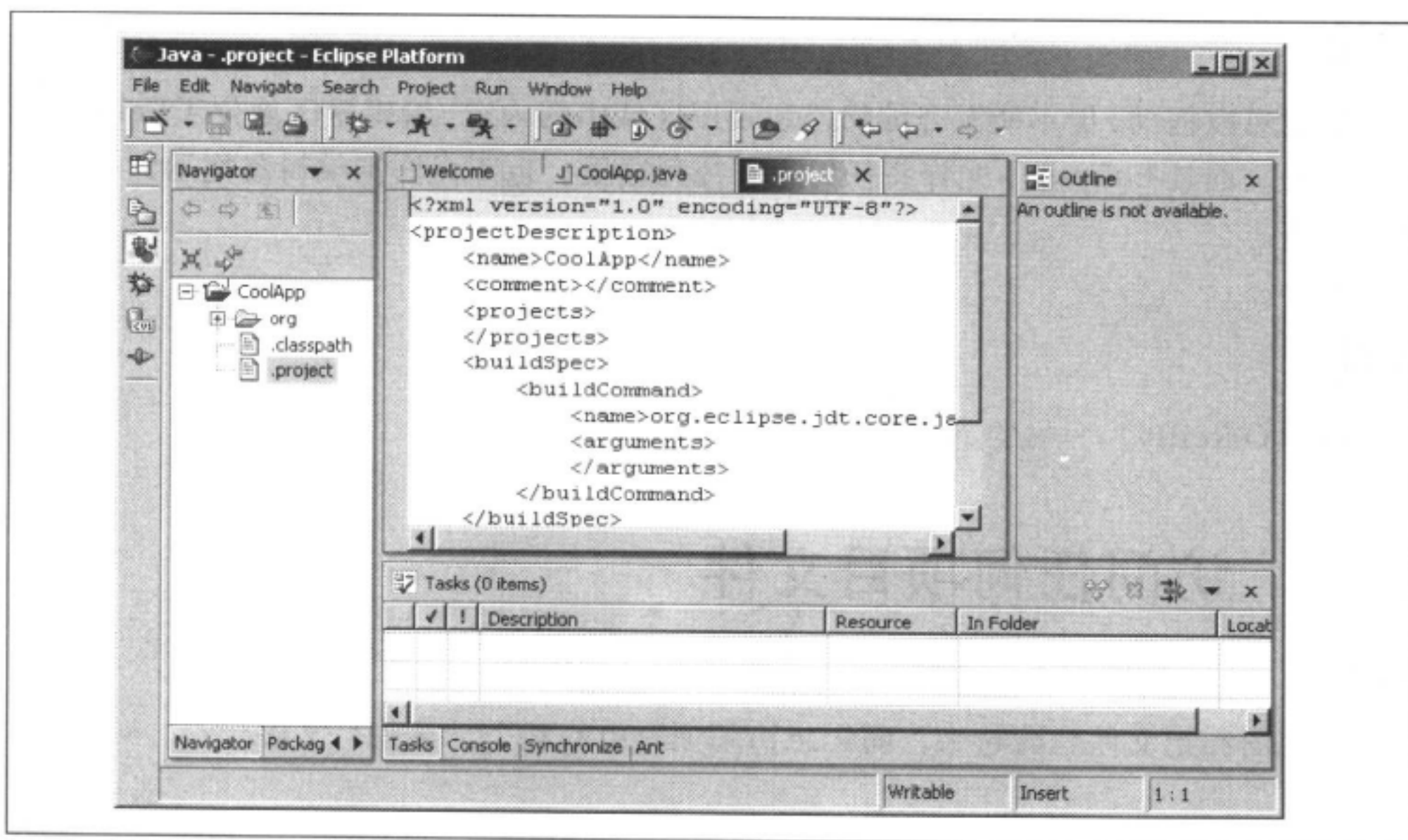


图 2-4：使用 Resource 透视图

解决方案

可以拖动编辑器，并在编辑器区域平铺它们。

讨论

当拖动一个编辑器，并把它放置在编辑器区域某个边缘附近时，可以看到一个箭头，如图 2-5 所示，表示放下编辑器时它将停靠在边缘处。

新的、平铺后的编辑器外观如图 2-6 所示。

只需将编辑器拖回到其原来的位置，即可使编辑器还原。

注意： 拖动视图（如 Package Explorer 视图）中的项目，并把它放到编辑器区域，即可打开该项目。

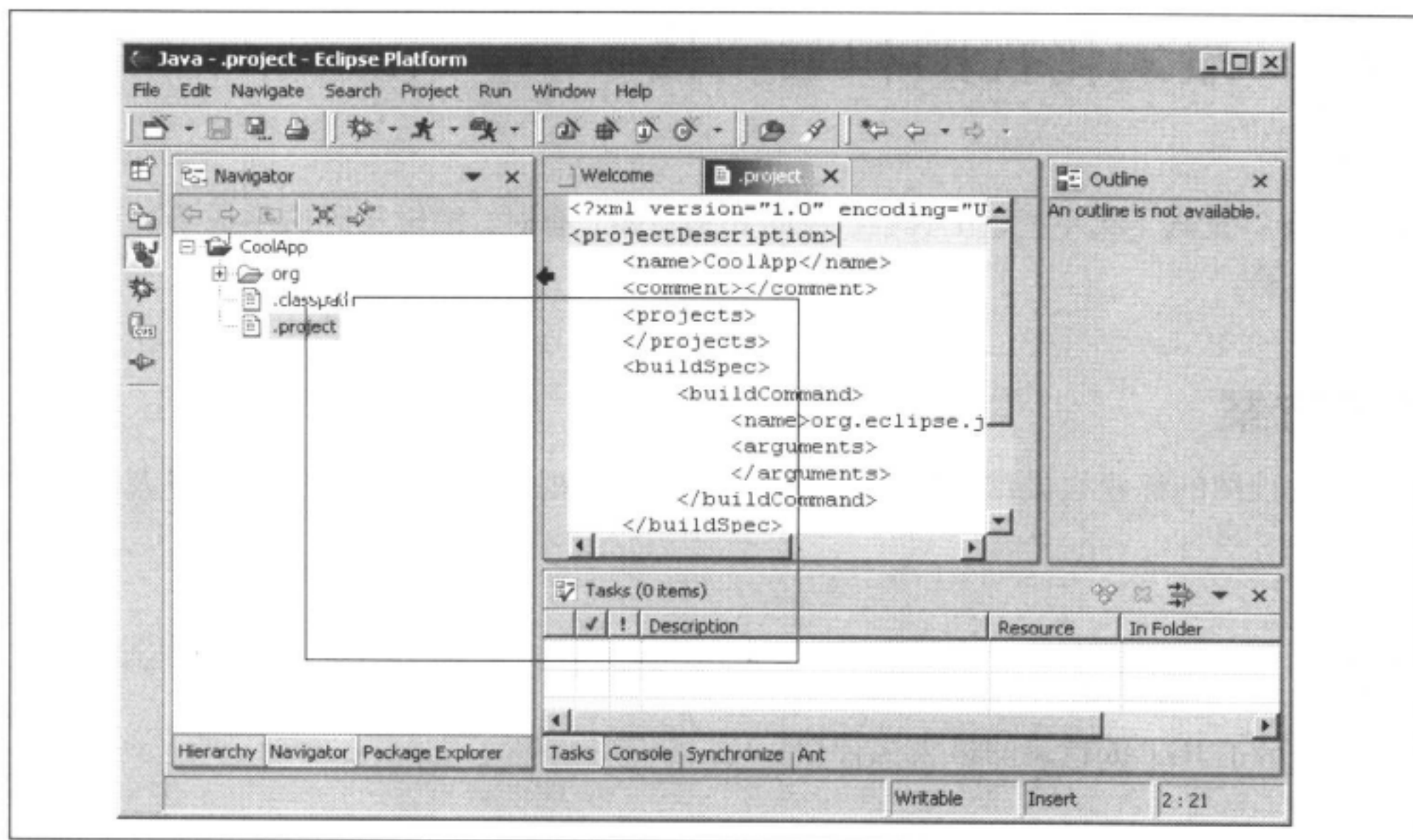


图 2-5: 拖动编辑器

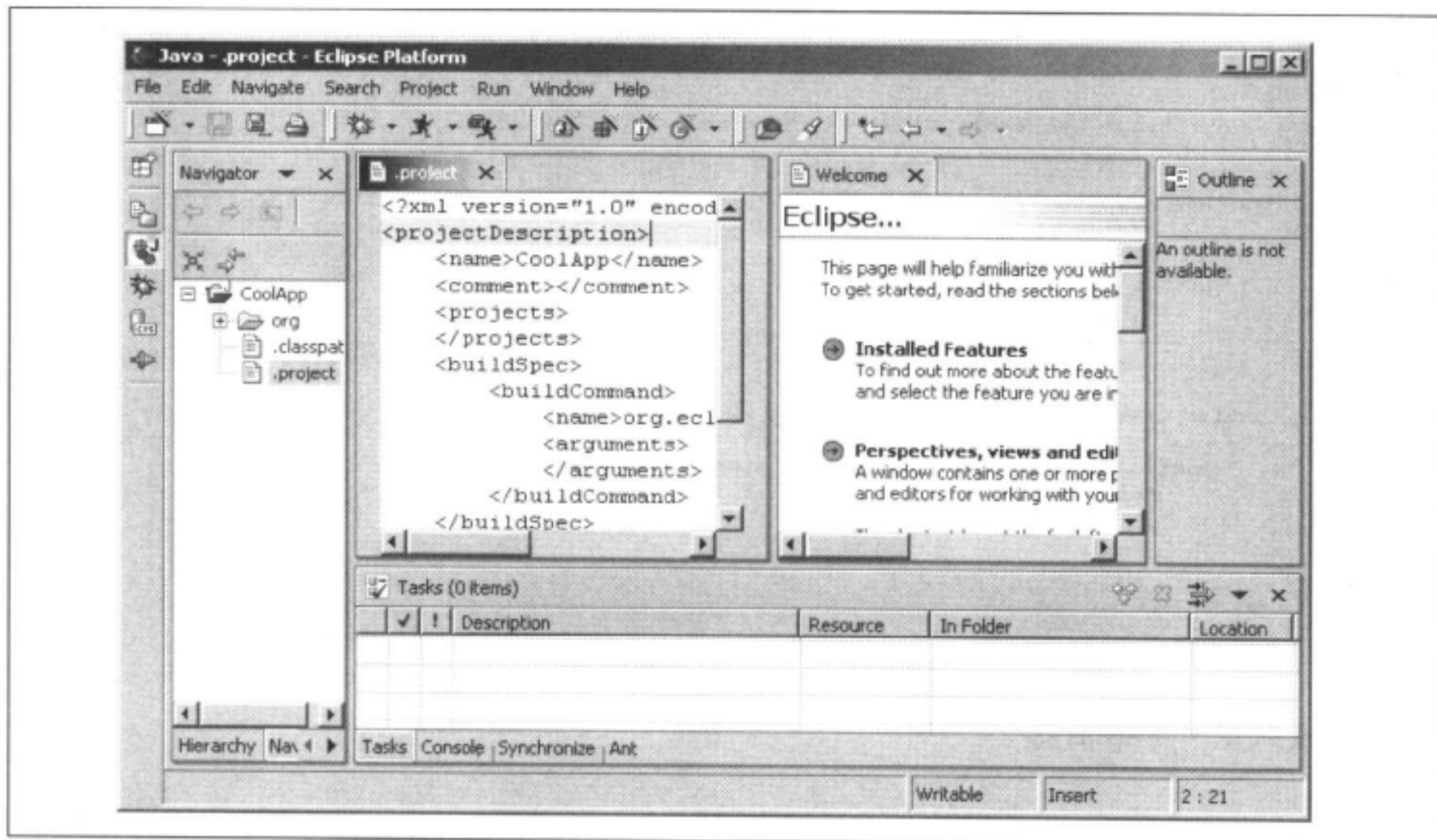


图 2-6: 平铺编辑器

2.5 最大化视图和编辑器

问题

Eclipse 窗口充满了菜单栏、工具栏、视图和编辑器。有时，编辑代码似乎成为一种束手束脚的体验。

解决方案

通过双击视图的标题栏或编辑器的标签，即可最大化视图或编辑器，但很少有人知道这一点。

讨论

图 2-7 显示了 JDT 编辑器的最大化视图。

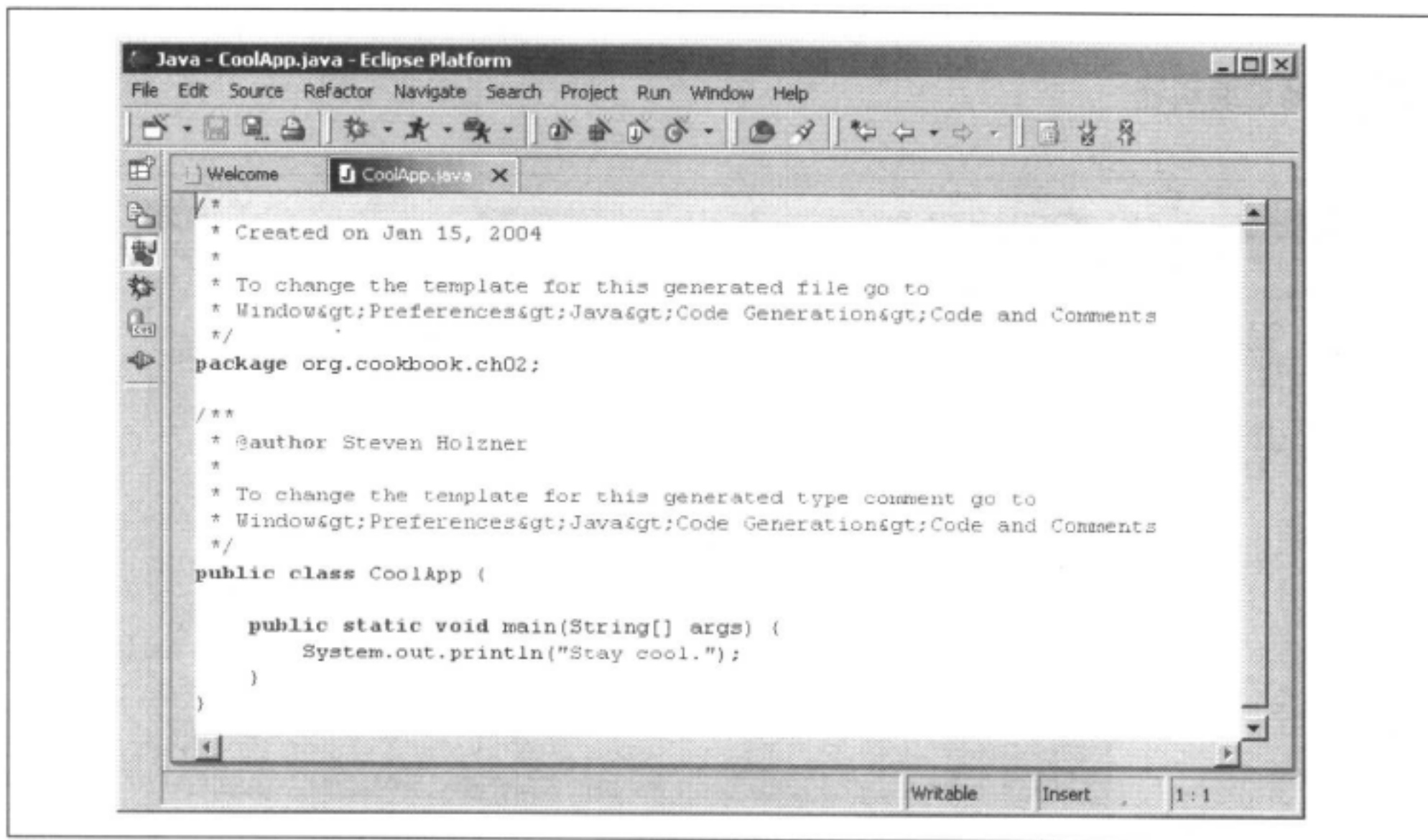


图 2-7：最大化一个编辑器

注意： 双击视图的标题栏或编辑器的标签，可以将其恢复到原来的大小。

与许多 IDE 一样，Eclipse 的窗口有时会变得拥挤不堪，特别是使用大型文档时。在编辑

代码时，你可能不想让这么多的东西来分散你注意力。要解决这一问题，双击编辑器的标签将其最大化即可。

2.6 返回上一个编辑器

问题

你已经打开了 20 个编辑器，而通过滚动这些编辑器的标签在其中进行导航，很费时间。

解决方案

使用工作台编辑器的导航历史记录，它与 Web 浏览器的历史记录非常相似。与在 Web 浏览器中的操作一样，单击后退按钮即可返回到上一个编辑器（也可以使用 Navigate 菜单的命令，如 Navigate → Backward，或使用键盘快捷键，如 Alt- 左箭头键）。

讨论

当使用 Eclipse 开发长期项目时，会发现编辑器区域充满了编辑器。而且，随着编辑器变得越来越拥挤，Eclipse 使编辑器的标签变得越来越小，且标签上的文字更加简短。打开的编辑器越多，在编辑器之间切换就越困难。解决这一问题的一种方法就是，牢记可以使用导航控件在最近使用过的编辑器之间切换。

2.7 返回到上一个编辑位置

问题

在编辑代码时，你切换到另一个编辑器，以确保变量名是正确的。但你打开了 15 个编辑器。如何准确地返回到刚才编辑的代码行呢？

解决方案

选择 Navigate → Go to Last Edit Location (Ctrl-Q)，即可返回到在编辑器中最后一次进行修改的位置。在使用编辑器时，可以看到工具栏上有与该功能对应的按钮。

讨论

Eclipse 使程序员可以方便地在编辑位置之间移动，对于任何一名程序员的工具集来说，

这种功能都是一种有益的补充。如果在当前透视图中看不到这些按钮，可能需要通过选择 Window → Customize Perspective → Other → Editor Navigation 来添加这些按钮。

2.8 将视图链接至编辑器

问题

在一个视图中打开的项目与编辑器之间缺乏密切的联系。默认情况下，切换编辑器，甚至关闭一个编辑器，都不能改变在视图中所做的选择。

解决方案

单击 Link with Editor 按钮。

讨论

许多视图，如 Resource 透视图的 Navigator 视图和 Java 透视图的 Package Explorer 视图，都可以与编辑器同步。要进行同步，单击 Link with Editor 按钮即可，如图 2-8 所示。

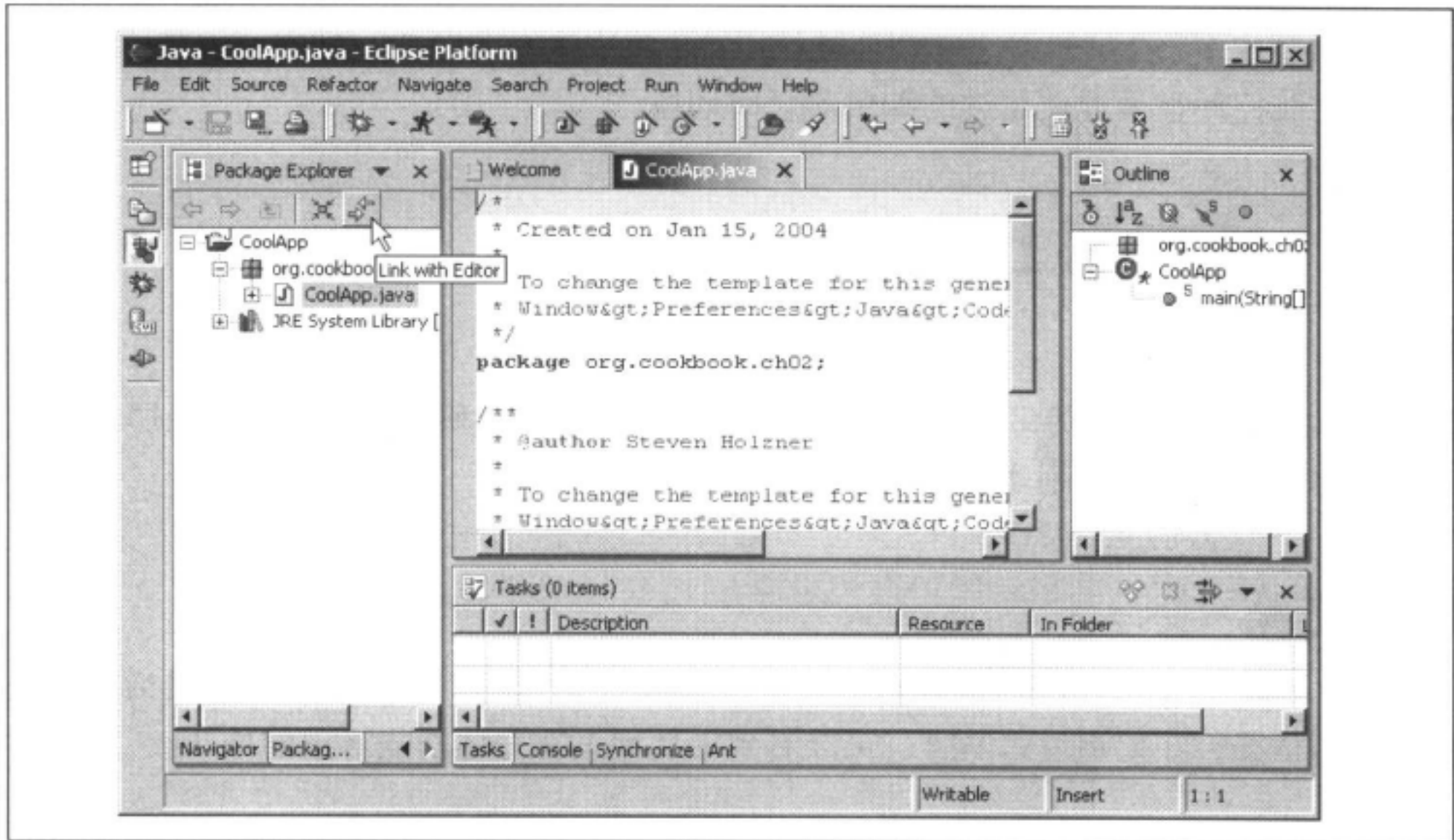


图 2-8: Link with Editor 按钮

通过将视图与编辑器链接，在视图和编辑器之间建立了一种联系，使得视图总是显示当前正在编辑的文件。

参考

Eclipse (O'Reilly) 一书的第 1 章。

2.9 重新排序视图和编辑器标签

问题

你已经打开了十多个编辑器,而目前正使用的那两个编辑器位于标签组的另一端。结果,为了使用所需的那两个编辑器,必须在全部编辑器标签之间来回滚动。有没有更好的办法?

解决方案

在 Eclipse 中,可以根据需要重新排列视图和编辑器标签的顺序。只需将标签拖至需要的位置即可。

讨论

这是一个十分有用的功能,但很少有 Eclipse 程序员知道,甚至那些每天都使用 Eclipse 的程序员也不知道。当有大量编辑器打开时,与同时处理多个项目时的情形一样,必须在编辑器标签之间来回滚动,才能找到所需的编辑器。为了避免这种情况发生,可以拖动编辑器标签,并放置在需要的位置,以重新组织当前正在使用的编辑器。

2.10 从编辑器导航到视图

问题

从视图中的一个文件导航到显示该文件的编辑器,是一件十分简单的事情。只需在视图中双击该文件即可。然而,如果你想返回,或者想从显示文件的编辑器导航到显示该文件的视图,该怎么办?

解决方案

选择 `Navigate` → `Show In`, 然后从子菜单中显示的视图列表中选择所需要的视图。

讨论

例如，如果你正在编辑一个 Java 文件，选择 **Navigate** → **Show In**，将打开 **Package Explorer** 视图。而且，你正在编辑的那个文件已被选中。

2.11 指定组合键

问题

你正在执行一项特定的任务，比如添加一个书签，或者向任务列表中添加任务，等等。你想为这样的任务指定一个简单的组合键。

解决方案

通过给 Eclipse 任务添加组合键，可以自定义 Eclipse。

讨论

要弄清哪些任务是可用的，可选择 **Window** → **Preferences** → **Workbench** → **Keys**，然后选择需要自动执行的任务，如添加一个书签。然后选择要与该任务关联的键的顺序，如图 2-9 所示。

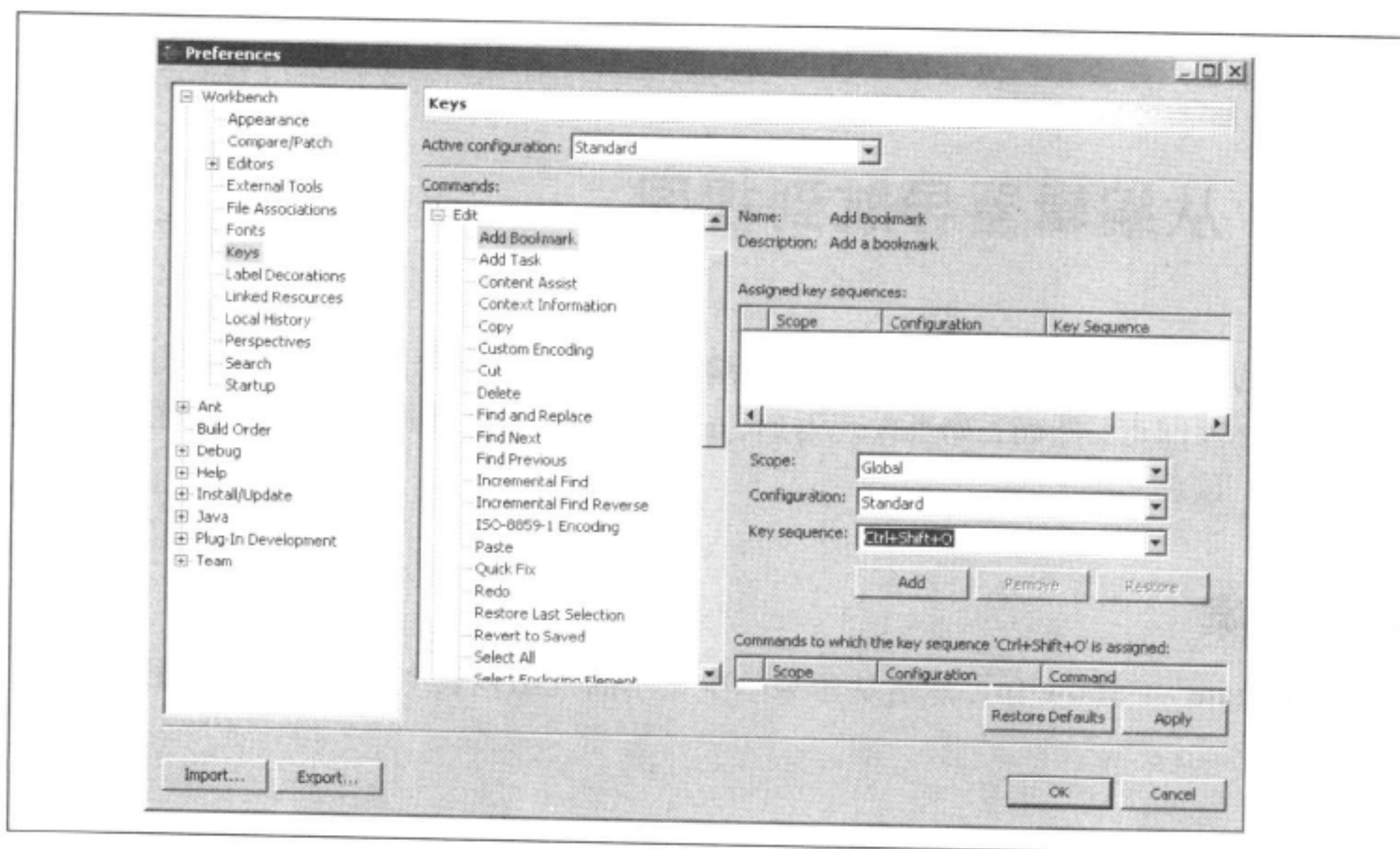


图 2-9：给任务指定组合键

2.12 通过图标显示更多的资源信息

问题

你想打开标签修饰器，以获取关于图标和按钮的进一步信息。

解决方案

选择 Window → Preferences → Workbench → Label Decorations。

讨论

标签修饰增加了个中视图中显示的标准Eclipse图标。例如，如果把代码文件归档到CVS库中，并打开了标签修饰，该文件的图标上将出现一个小的金色圆柱，而其他文件则没有。

要打开标签修饰功能，可选择Window → Preferences → Workbench → Label Decorations，然后选择需要使用的修饰效果，如图 2-10 所示。

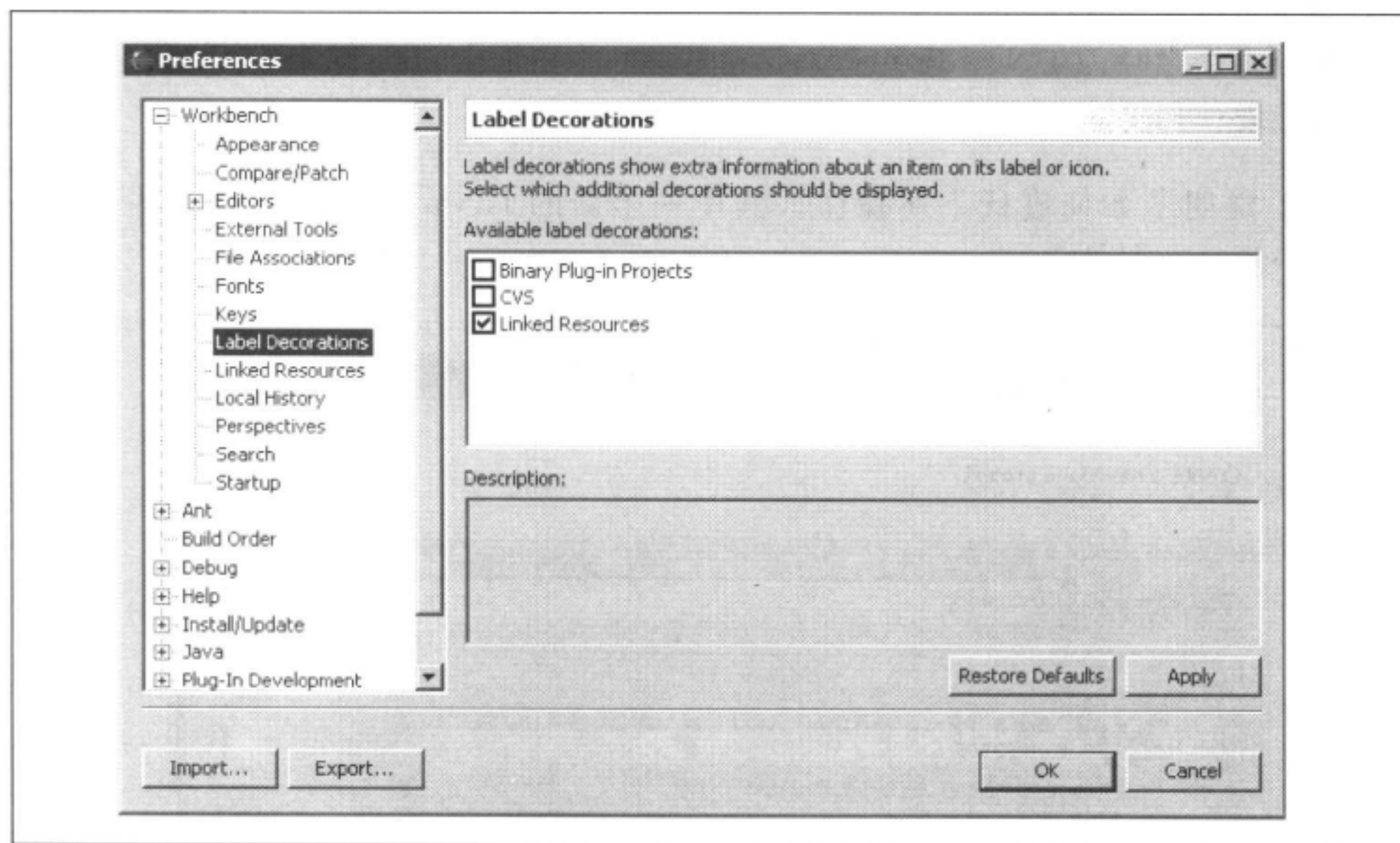


图 2-10：选择标签修饰

本书第 6 章介绍 Eclipse 与 CVS 的内容时，将使用标签修饰。一旦打开了标签修饰，一眼就可以看出哪些文件存储在 CVS 库中。

参考

6.6 节，以可视化方式标记版本控制下的文件。

2.13 使用不同的工作区

问题

你不想在一个 Eclipse 项目中使用默认的工作区。

解决方案

为项目命名时，在 New Project 对话框的第一个画面中，取消 Use default 复选框，并填上要使用的目录。

讨论

默认情况下，在创建新的 Eclipse 项目时，Eclipse 将新项目存储在其自己的工作区目录中。但也可以使用其他的目录，与所创建的 Eclipse 项目的类型无关。要选择不同的位置，为项目命名时，在 New Project 对话框的第一个画面中，取消 Use default 复选框。然后填上要使用的目录。

图 2-11 说明了如何进行上述操作，其中一个新的 Java 项目被创建，并存储在 `c:\myworkspace` 目录中。

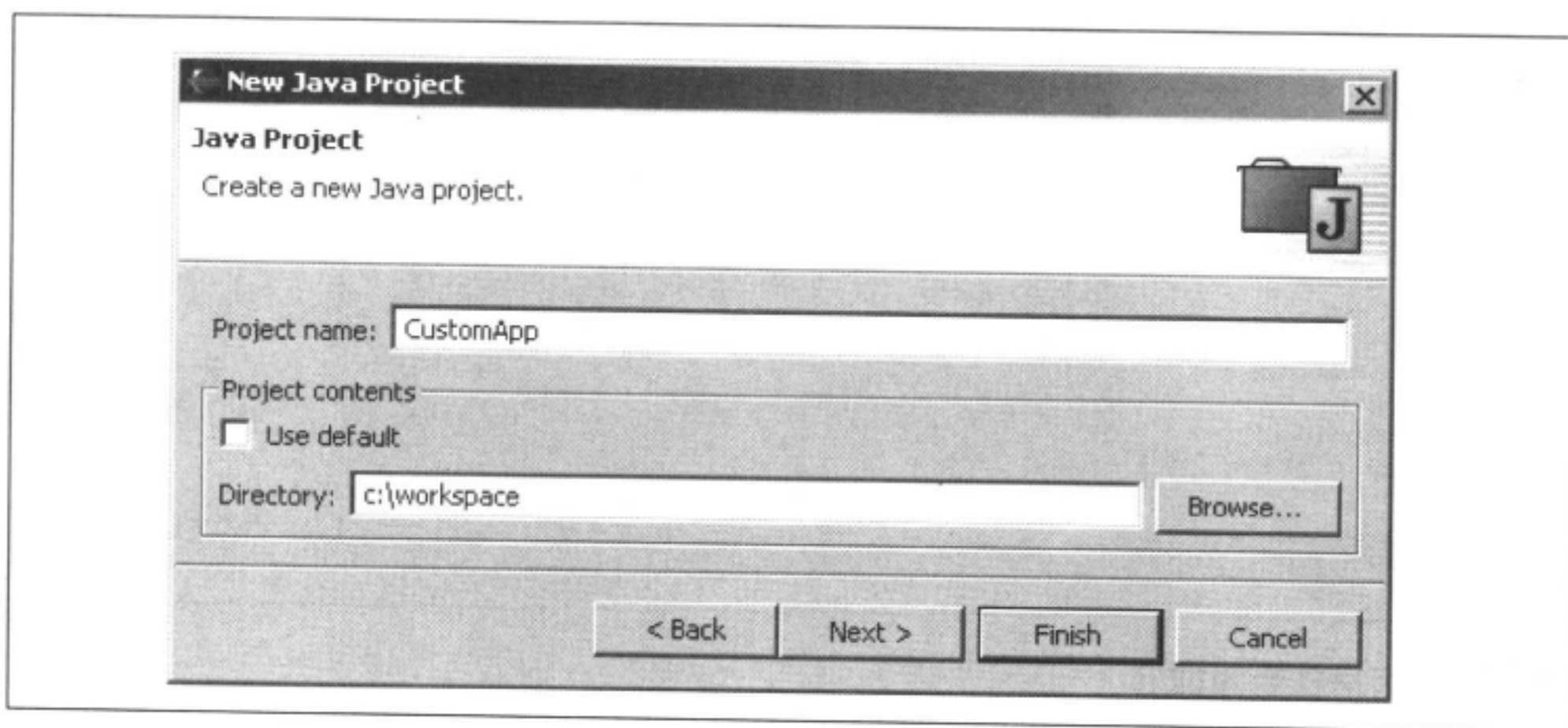


图 2-11：创建自定义的工作区

在许多情况下，适合使用非默认的工作区。例如，你可能正在通过网络处理代码。在这种情况下，应该将代码存储在其他软件能够使用的地方（在第 8 章中，我们将开发代码并将其存储在 Tomcat Web 服务器能够找到的地方，这意味着将使用 Tomcat 安装的 *webapps* 目录作为工作区）。当需要保存某个软件，比如正在开发的软件的测试版，以便与该软件的当前版分开时，使用非默认的工作区是一个好主意。

2.14 创建任务

问题

Tasks 视图中的 Eclipse 任务用作待完成工作的提示。例如，Tasks 视图中出现了编译错误时，可以通过单击这些错误，跳转到包含这些错误的代码行。那么，如何在 Tasks 视图中添加和管理自己的任务呢？

解决方案

在 Tasks 视图中右击，然后选择 New Task，即可创建一个新任务。这将打开 New Task 对话框，通过此对话框可以创建任务。

讨论

New Task 对话框如图 2-12 所示。输入新任务的名称，选择一个优先级，然后单击 OK。

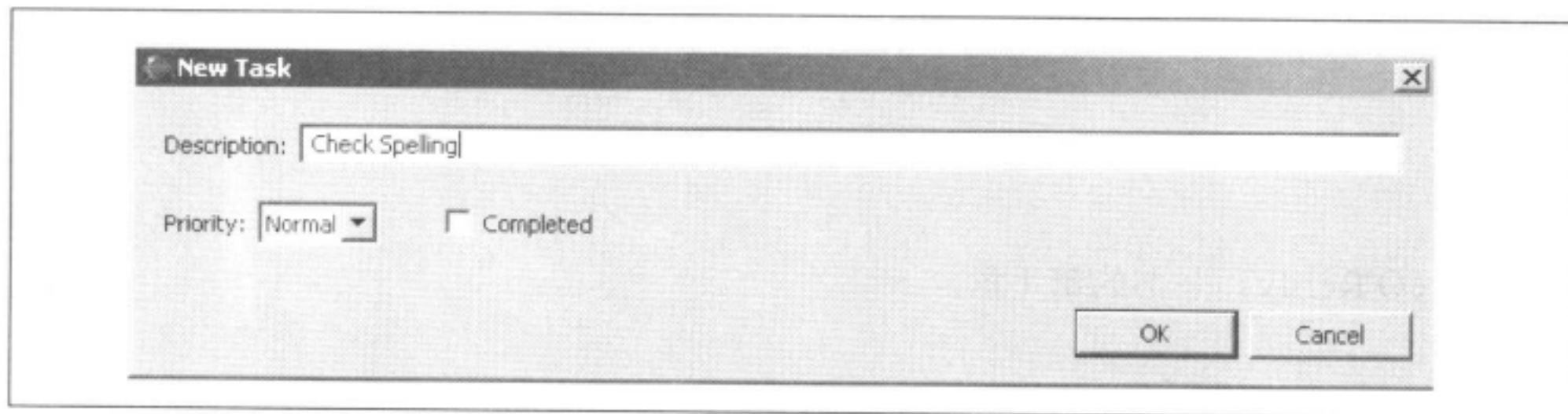


图 2-12：创建一个新任务

注意：通过右击编辑器的标记栏，然后选择 Add Task，或者单击 Tasks 视图工具栏上的带有 3 个加号的按钮，也可以创建新任务。

新建的任务将出现在 Tasks 视图中，如图 2-13 所示。

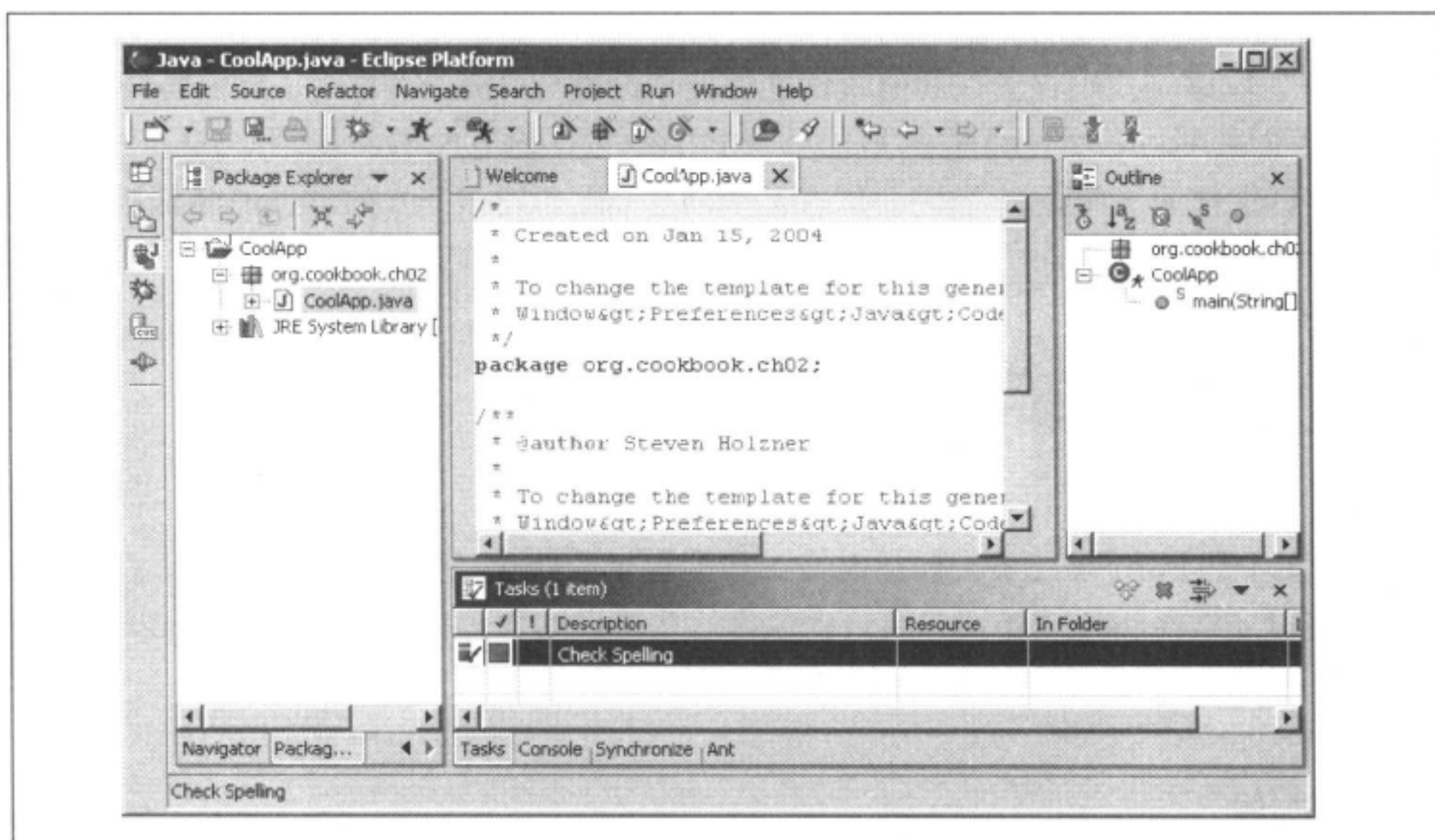


图 2-13: 新建任务

通过右击任务，然后单击 Delete 按钮，或者选中任务并按 Delete 键，可以删除添加到 Tasks 视图中的任务。

任务还有一个完成状态，通过右击任务，然后单击 Mark Completed 可以设置任务的完成状态。如果想删除所有已完成的任务，可右击 Tasks 视图，然后单击 Delete Completed Tasks。

参考

Eclipse (O'Reilly) 一书的第 1 章。

2.15 创建书签

问题

你想在以后返回到文件中的某个位置。

解决方案

通过右击编辑器的标记栏，然后在 New Bookmark 对话框中指定书签的名称，即可创建书签。

讨论

你正在编辑代码，忽然发现到了下班时间。如何保存编辑的位置呢？创建一个书签，以后在 Bookmarks 视图中双击该书签，即可回到原来的位置。右击标记栏，然后在 New Bookmark 对话框中命名书签，即可创建书签。书签将出现在标记栏上，如图 2-14 所示。

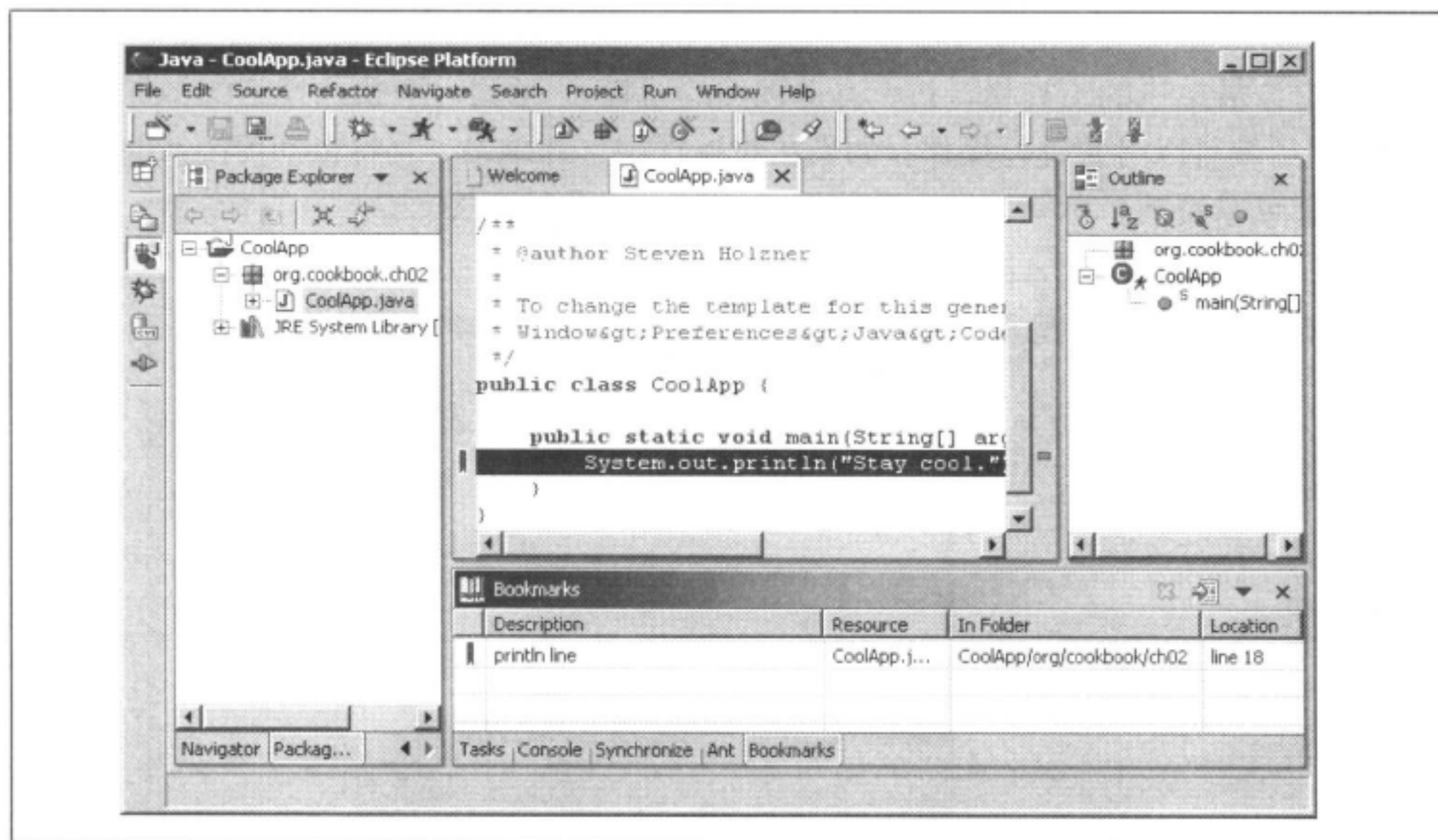


图 2-14：使用书签

以后需要回到书签标记的位置时，通过选择 Window → Show View → Other → Basics → Bookmarks，打开 Bookmarks 视图即可。在图 2-14 底部的 Bookmarks 视图中，可以看到所创建的书签；双击该书签即可在编辑器中打开标记的代码行。

不出所料，书签发生了作用，即使含书签的文件被关闭也一样；双击书签，文件将自动打开。

2.16 创建快速视图

问题

在 IDE 中，屏幕空间始终是稀缺的资源；视图可以重叠在一起，但不能打开太多的视图，否则很难找到正确的视图标签。

解决方案

使用快速视图可以释放 Eclipse 窗口中的一些空间。

讨论

将一个视图转换为一个快速视图后，它将以图标的形式添加到快捷工具栏上（即位于 Eclipse 窗口最左边的工具栏，其中有透视图的图标）。单击一个快速视图的图标，使其弹出，然后就可以对其进行操作。如果在快速视图之外单击，它将自动关闭。例如，在图 2-15 中可以看到，Bookmarks 视图被转换成一个快速视图。

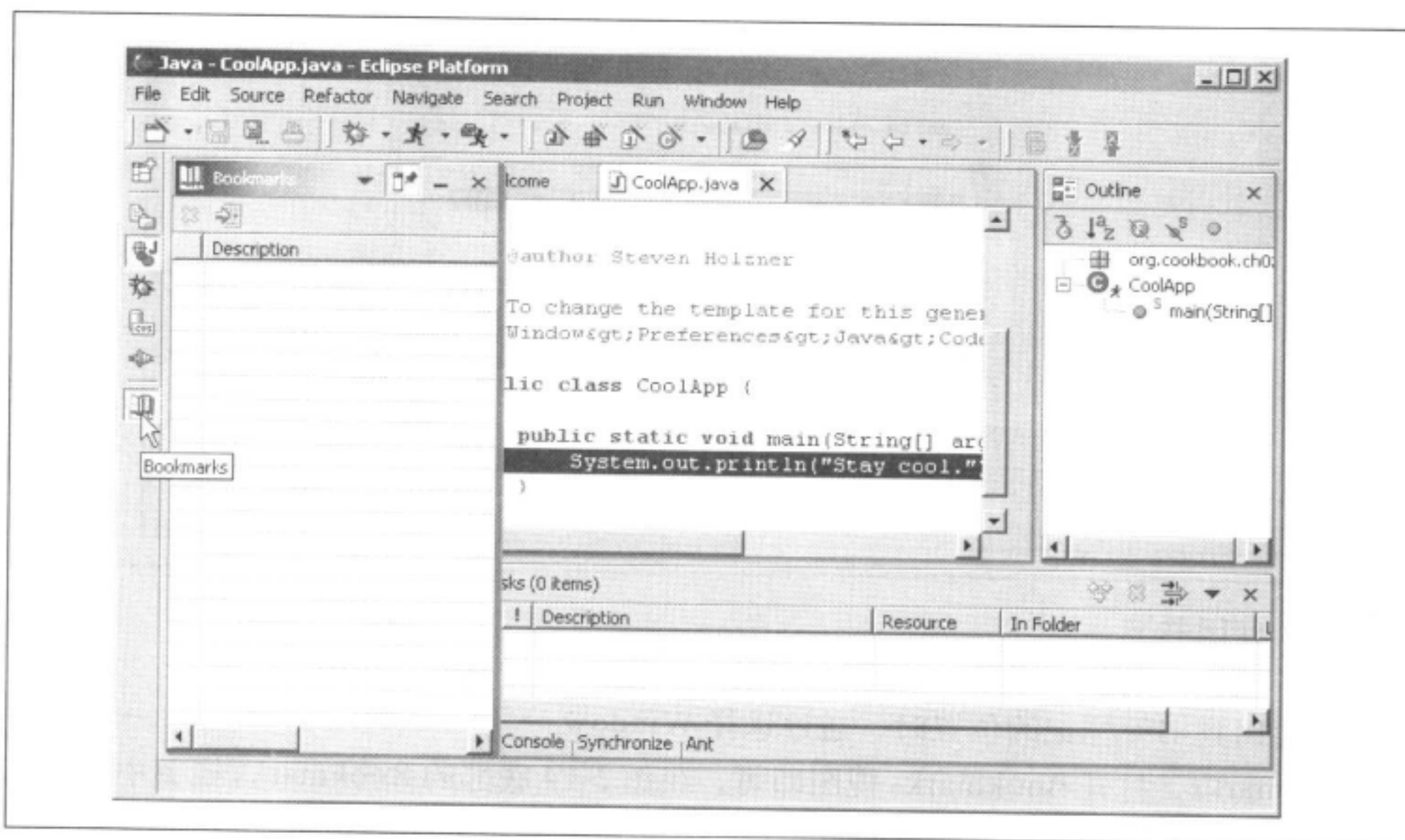


图 2-15：创建一个快速视图

右击视图标题栏上的视图名称，打开视图的系统菜单，从中选择 Fast View，即可将一个视图转成快速视图。当把一个普通视图转换成快速视图时，该快速视图的图标将出现在快捷工具栏上。右击快捷工具栏上的快速视图图标，并取消选择 Fast View，可以将快速视图还原。

2.17 自定义帮助

问题

你正在浏览 Eclipse 的帮助系统，其中一些帮助页面十分的长。你想搜索这些长长的页面上的文字，但你发现，Eclipse 帮助系统的浏览器不允许搜索当前页。

解决方案

使用自己的浏览器。

讨论

Eclipse 使用 Tomcat Web 服务器的自定义版本来显示帮助文档。当帮助文档打开时，可以直接找到要查看的帮助主题的 URL。在启动了你自己的浏览器以后，可以导航到该 URL。

要获得一个帮助主题的 URL，可以右击该主题，然后选择 Create Shortcut，这将创建该帮助主题的快捷方式。例如，我的关于书签的帮助主题的 URL 为 `http://127.0.0.1:1203/help/index.jsp?topic=/org.eclipse.platform.doc.user/concepts/cbookmrk.htm`。双击该快捷方式，或者在浏览器中输入该 URL，即可在浏览器中查看该帮助页面，如图 2-16 所示。可以用浏览器的“编辑”菜单，自由地搜索 Eclipse 的帮助页面。

注意： Eclipse 的最新修订允许在 Eclipse 的帮助系统中为帮助主题添加书签。要创建书签，可单击工具栏上的 Bookmark Document 按钮，书签将出现在 Bookmarks 页上。双击一个书签，即可回到标记的帮助主题。

甚至可以自定义 Eclipse 的帮助系统，以使用你的默认浏览器。只需选择 Window → Preferences → Help，在帮助页面选择所需的浏览器即可，如图 2-17 所示。

注意： 如果你不想等待帮助系统打开，该怎么办？实际上，要等待的是 Tomcat Web 服务器启动，以便将帮助文档发送给 Eclipse 的 Help 浏览器的时间。如果只是想查一下一些帮助主题是否符合需要，可单击 Eclipse 工具栏上的 Search 按钮和 Search 对话框中的 Help Search 按钮。输入有关帮助主题的文本，单击 Search 按钮即可。Search 视图将显示帮助系统中匹配的搜索结果。在此视图中，可以查看搜索结果是否与查询的主题相匹配，所有操作均无需启动 Tomcat 服务器。然而，如果想查看感兴趣的搜索结果，必须启动帮助系统的浏览器，才能查看该主题的具体内容。

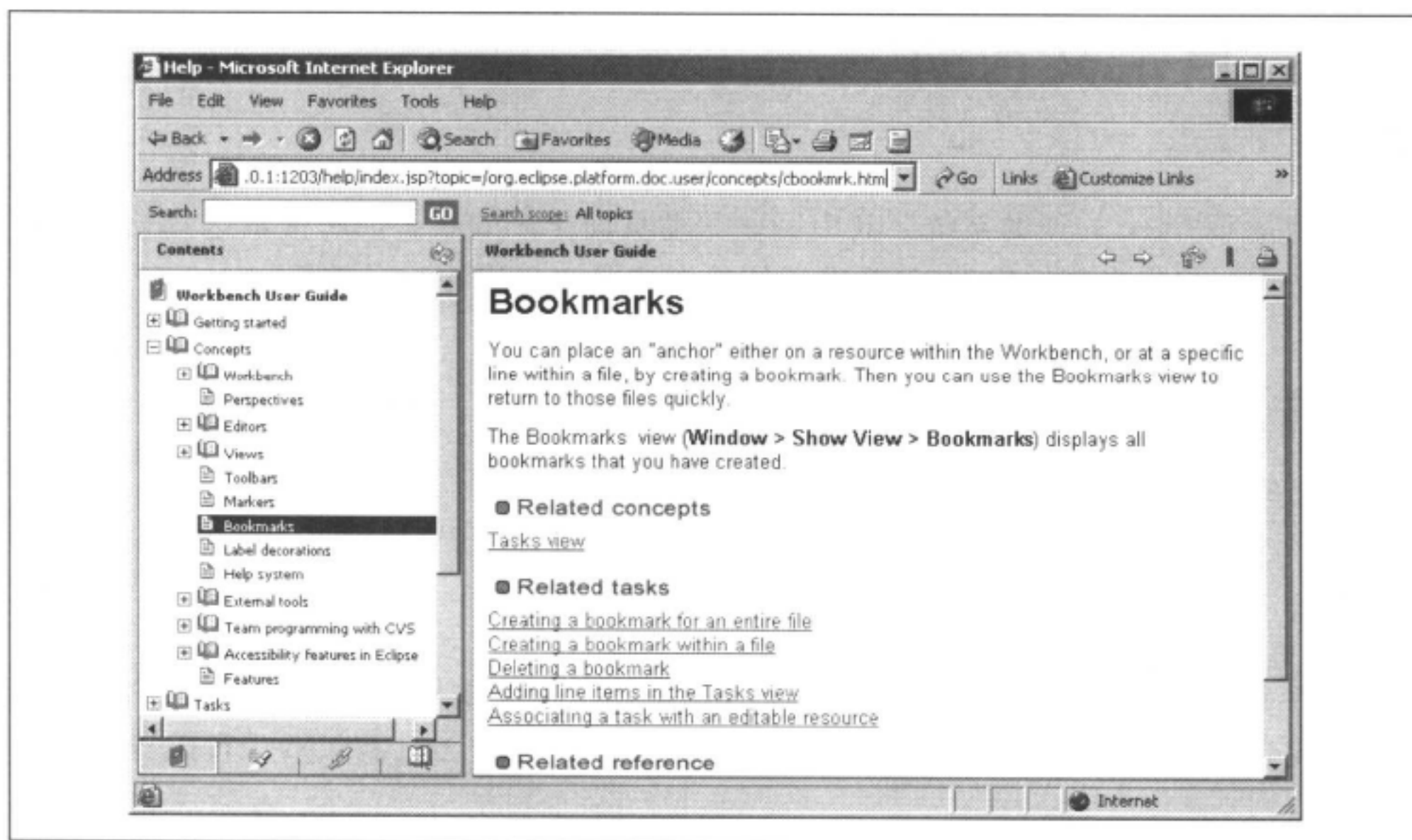


图 2-16: 在自己的浏览器中打开 Eclipse 的帮助文档

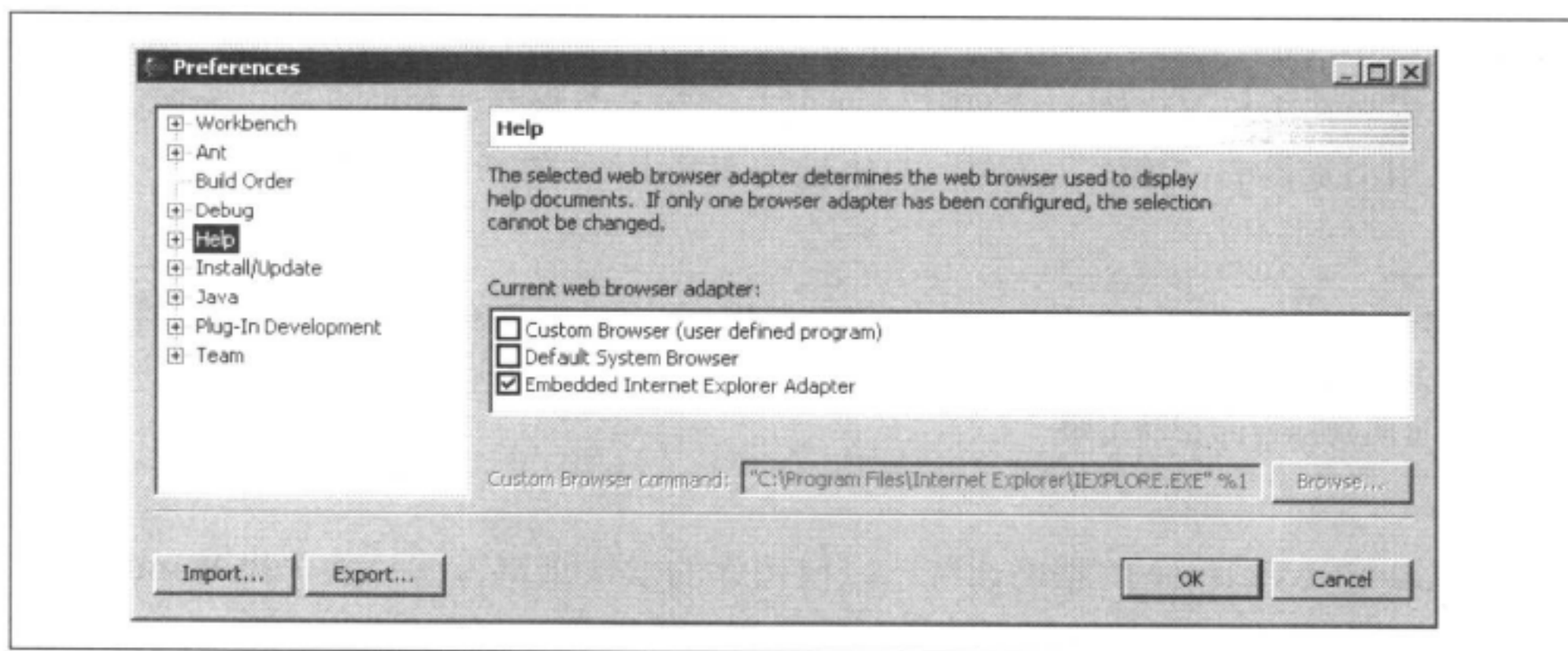


图 2-17: 配置帮助浏览器

2.18 恢复删除的资源问题

一个必需的文件被误删除，而你想恢复它。

解决方案

没有问题。使用容器项目的 Restore 视图，可以从 Local History 快捷菜单项中恢复文件。

讨论

要恢复已删除的文件，可右击文件所属的项目，从 Local History 快捷菜单中选择 Restore，打开如图 2-18 所示的对话框。在此图中，恢复了删除的一个 `.class` 文件。

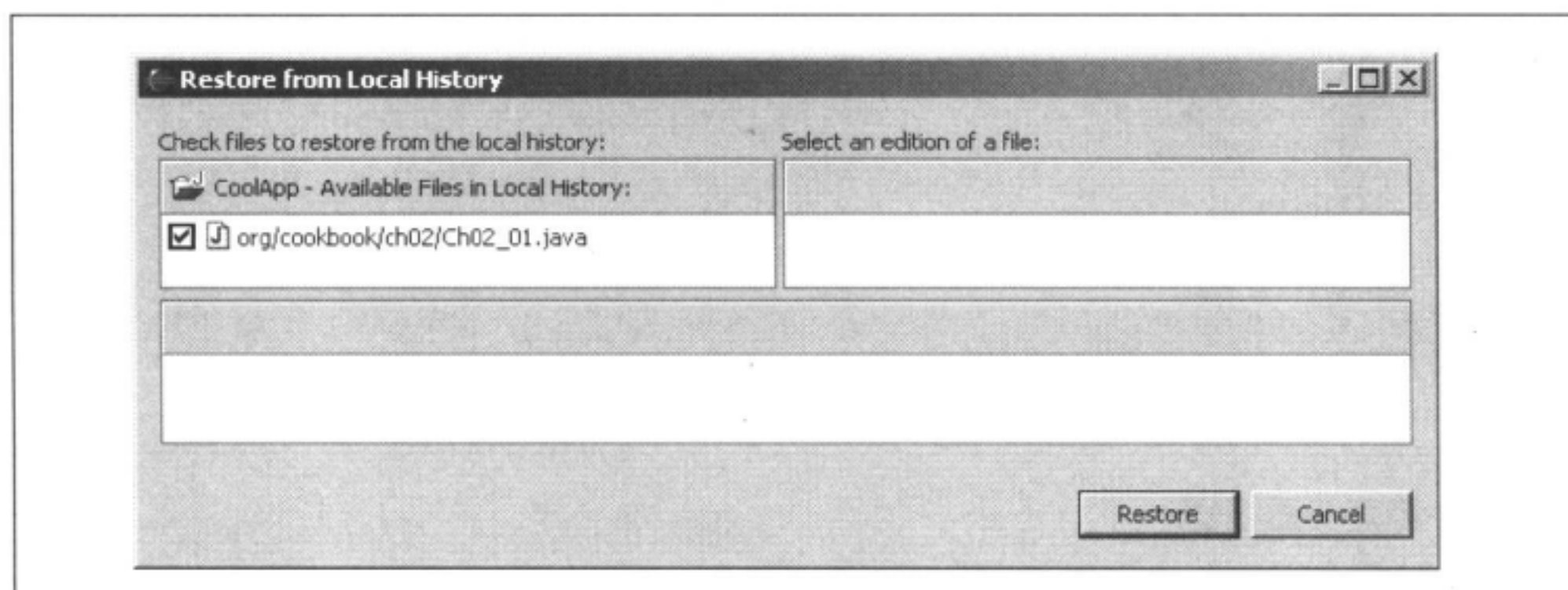


图 2-18：恢复删除的文件

注意： 由于 Java 透视图的 Package Explorer 视图并不显示所有的文件，包括 `.class` 文件，所以，如果要恢复 `.class` 文件，必须切换到 Java 透视图的 Navigator 视图，或使用 Resource 透视图。

2.19 自定义透视图

问题

你想改变菜单项和透视图中的工具栏项目。

解决方案

使用 Customize Perspective 对话框；选择 Window → Customize Perspective，可以打开此对话框。Customize Perspective 对话框如图 2-19 所示。

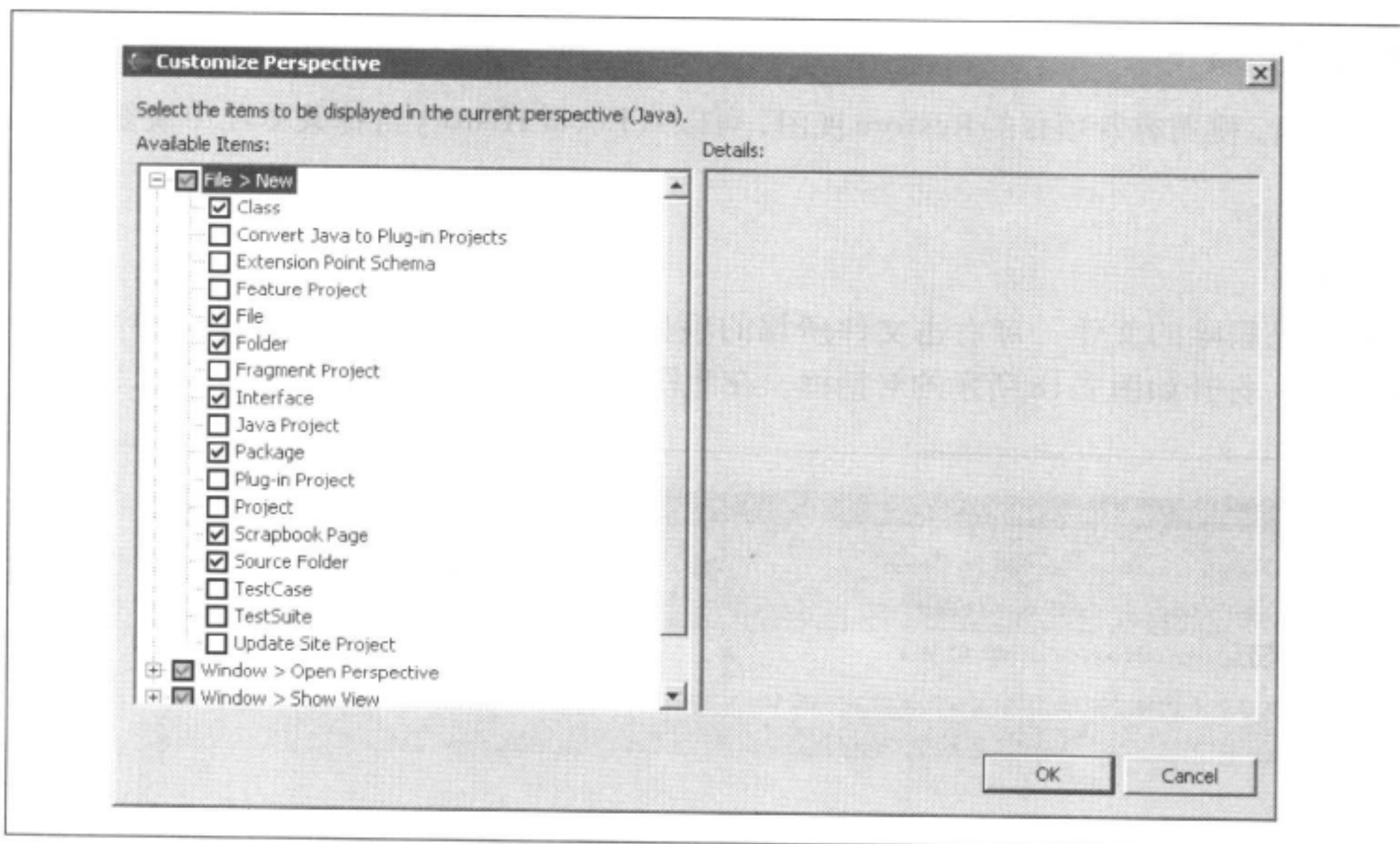


图 2-19：自定义透视图

注意： 在设置了Eclipse的参数之后，可以导出这些参数，使其他程序员也可以使用这些参数。为此，可单击 Preferences 对话框中的 Import 按钮和 Export 按钮。

注意： 通过Preferences对话框可以自定义菜单项和工具栏项目，但并不能真正使你自定义透视图（事实上，你可以添加的菜单项通常总能在Other菜单中找到）。要了解如何自定义透视图，请参阅 2.21 节。

参考

2.20 节，还原透视图；2.21 节，创建新透视图。

2.20 还原透视图

问题

Eclipse的一个透视图曾经重新排列，而且有太多的视图被关闭。如何把透视图还原到其初始状态？

解决方案

如果一个透视图变得杂乱无章,以至于无法辨认,通过选择 Window → Reset Perspective, 可以将其还原到其初始状态。

讨论

Eclipse 总是在其内存中保存大量的备份和还原点。除了平台和文件特有的备份以外, Eclipse 还保存有编辑器与其他组件的窗口与透视图的默认起点。Reset Perspective 对话框提供只需鼠标单击的操作方式,当透视图变得杂乱无章时, Reset Perspective 对话框是非常有用的。

2.21 创建新的透视图

问题

内置的透视图都不是十分合用。你想对内置的透视图进行组合和匹配,以创建自定义的透视图。

解决方案

没问题! 只需打开一个与你想创建的透视图相近的透视图,添加新的视图,并关闭不需要的视图,然后选择 Window → Save Perspective As 保存新建的透视图即可。

讨论

Save Perspective As 对话框如图 2-20 所示。这个新建的、命名为 Debug2 的透视图,其实就是把 Navigator 视图添加到了 Debug 透视图。

现在,你可以随时打开新建的透视图,如图 2-21 所示。

注意: 若要删除自定义的透视图,可以选择 Window → Preferences → Workbench → Perspectives, 选中要删除的透视图,然后单击 Delete。

能够创建自己的透视图真是太好了,事实上,这是其他任何 IDE 都不具备的功能。在本书中,你将有机会组合和匹配视图,创建全新的透视图。创意无限!

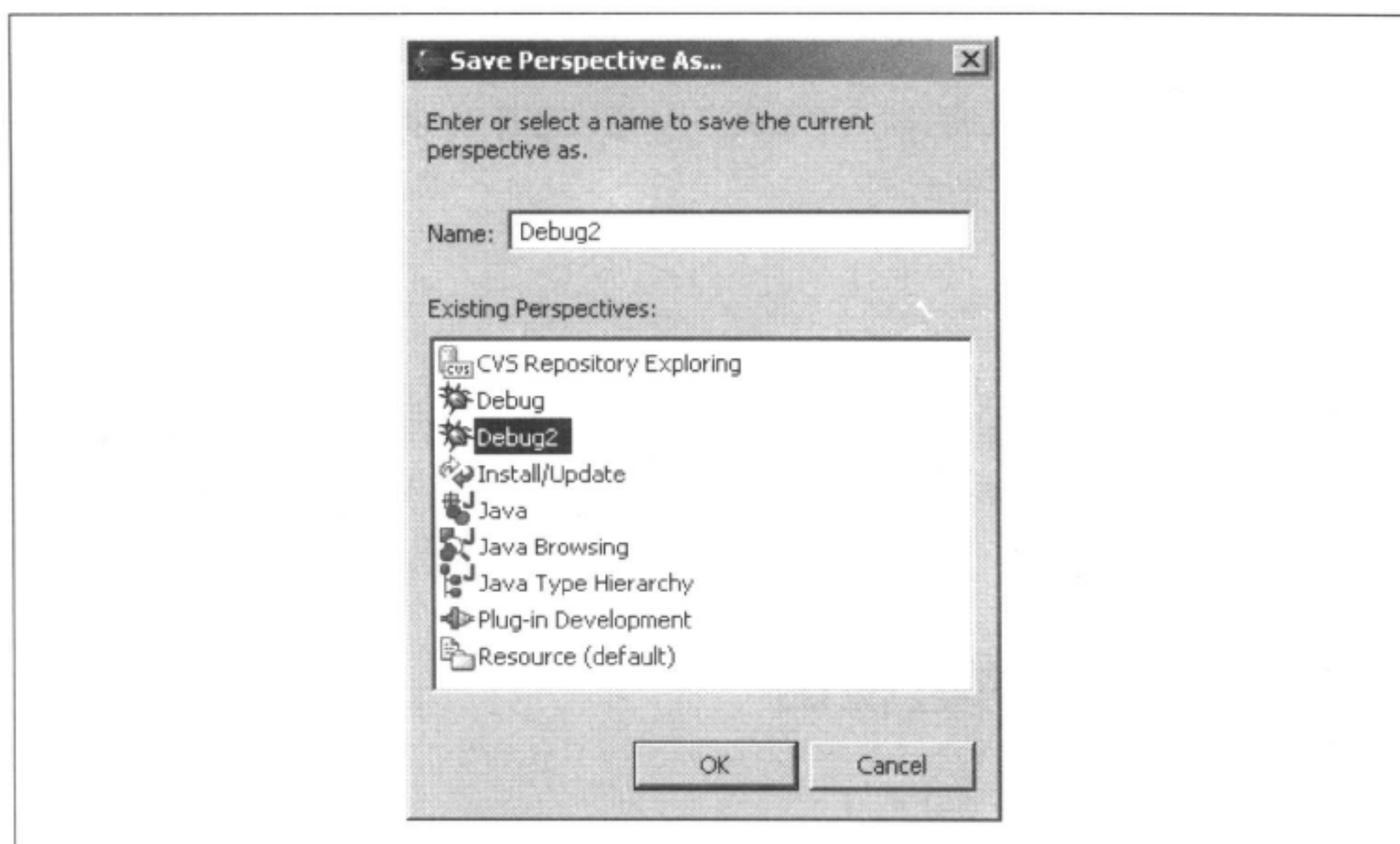


图 2-20: 创建一个新的透视图

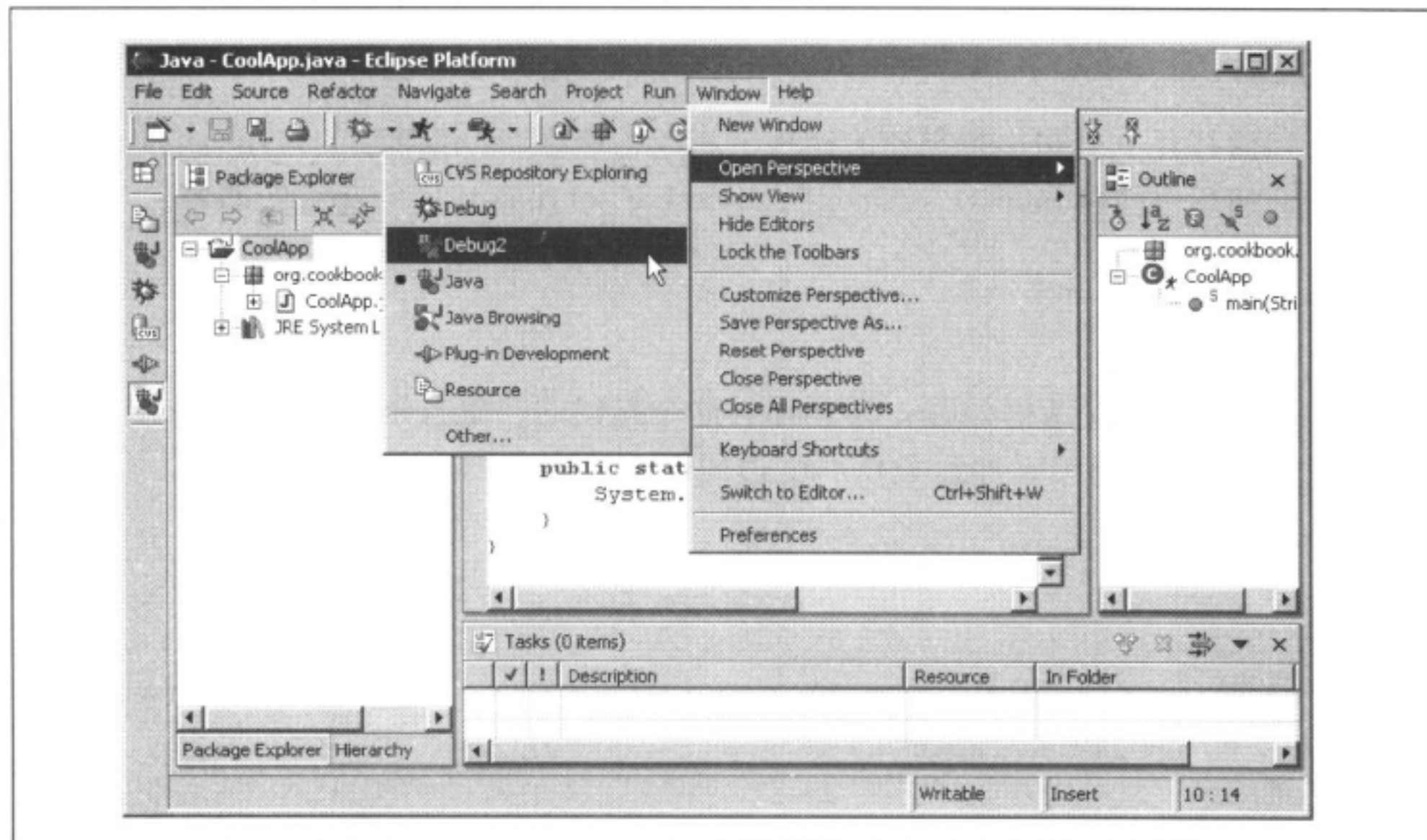


图 2-21: 打开新建的透视图

3.0 简介

绝大多数 Eclipse 开发人员都在开发 Java 代码。从本章开始，我们的注意力将转到 Java 开发上来。本章将介绍大量的基础知识，从加速 Eclipse JDT 以便更快地输入代码，到使用 JDT 创建 Java 项目、包、类和方法，等等。

Eclipse 开发团队一直在不停地思考可以用来促进 Java 代码开发的技术。本章将重点介绍其中的一些技术。我们将在第 4 章继续讨论 Eclipse JDT，即深入讨论重构、搜索、编译（building）和发布应用程序。

3.1 加速 JDT 编辑器

问题

在 JDT 编辑器中输入代码时，各种窗口突然出现，红色波浪线出现在文本下方，而红色方框出现在标尺上，还有其他一些令人分心的事情。

解决方案

可以关闭许多自动语法检查和问题检查的功能，以加快代码输入速度。

讨论

虽然 Eclipse 提供了许多自动语法检查和问题检查功能，但这些功能有时令人讨厌。幸运的是，Eclipse 的自定义几乎是无限限制的。下面是一些比较常用的、会令人分心的自动检查功能，包括对付这些功能的提示：

输入代码时出现问题指示器

通过选择 Window → Preferences → Java → Editor → Annotations 并取消所有复选框，可以关闭问题指示器。

总标尺

通过选择 Window → Preferences → Java → Editor → Appearance → Show Overview Ruler，然后取消相应的复选框，可以隐藏总标尺。

大纲同步

通过选择 Window → Preferences → Java → Editor → Appearance → Synchronize Outline Selection on Cursor Move，然后取消相应的复选框，可以关闭大纲同步功能。

Eclipse 3.0

智能插入模式是 Eclipse 3.0 中新增的另一项自动功能，可以将其打开或关闭。在这种模式下，当你输入代码时 Eclipse 可以自动完成一些事情，比如为字符串添加闭引号，添加闭花括号，等等。在 Eclipse 3.0 中，通过按下 Insert 键，可以打开或关闭智能插入模式；按 Insert 键将在改写模式和插入模式（Eclipse 3.0 与 Eclipse 2.x 都有这两种功能）以及 Eclipse 3.0 智能插入模式之间循环切换。在图 3-1 所示的 JDT 编辑器中，可以看到智能插入模式的光标，它看起来像左方括号。

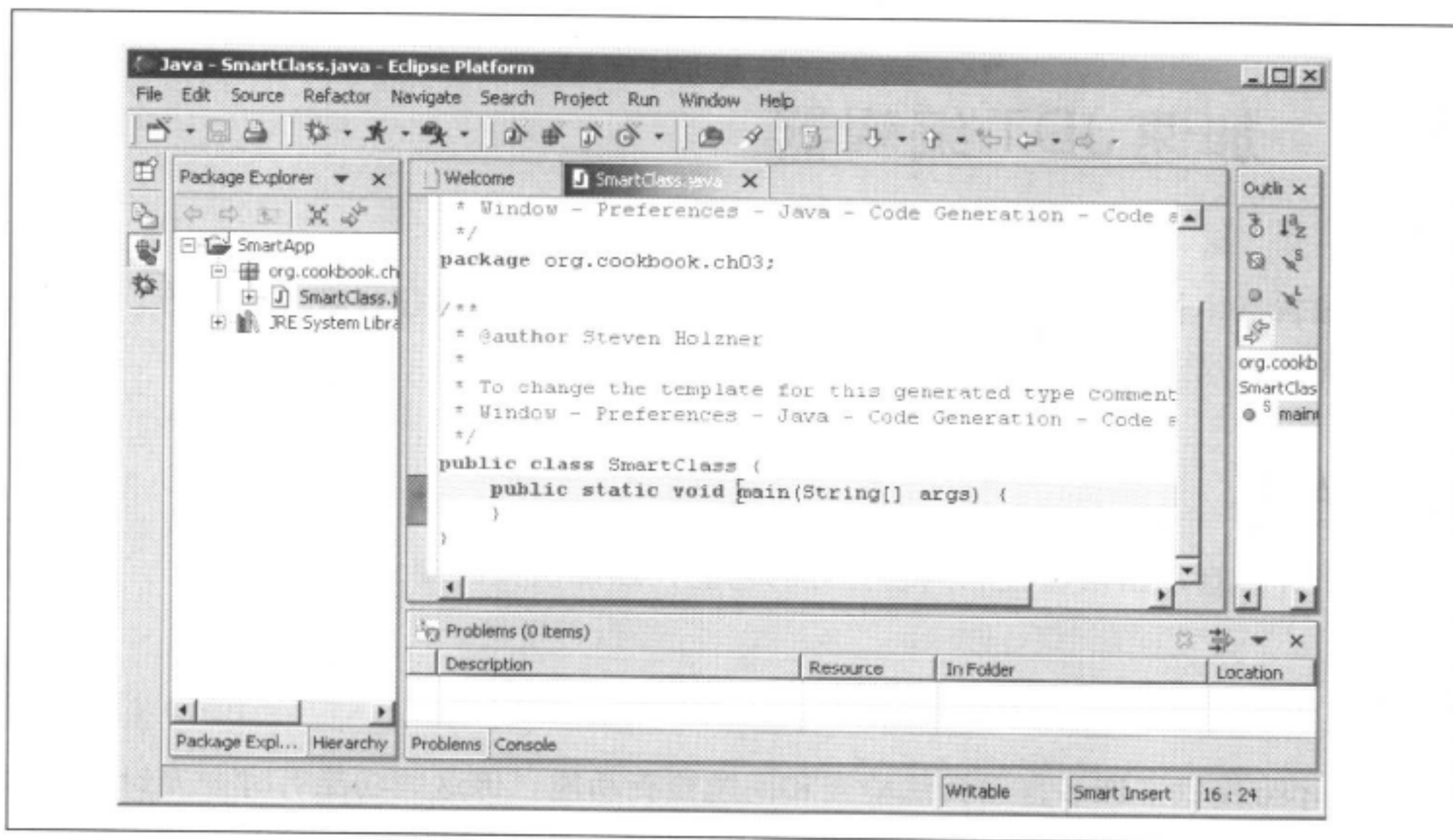


图 3-1：Eclipse 3.0 的智能插入模式

注意：要配置JDT编辑器是否自动合围字符串、添加花括号等，并配置Eclipse 3.0的智能插入模式，可选择 Window → Preferences → Java → Editor → Typing。

3.2 创建一个 Java 项目

问题

你想开始编程，所以需要新建一个 Java 项目。

解决方案

选择 File → New → Project，或者在 Java 透视图右击 Package Explorer 视图，并选择 New → Project。

讨论

在 Eclipse 中，所有 Java 代码都必须放在一个某种类型的项目中。虽然在第 1 章已经介绍了创建新项目的基本知识，而在本章我们将更深入地探讨这一过程。

要创建一个 Java 项目，可通过选择 File → New → Project，或通过右击 Package Explorer 视图，并选择 New → Project，打开 New Project 对话框。在左边的列表框中选择 Java，而在右边的列表框中选择 Java Project，如图 3-2 所示，然后单击 Next。

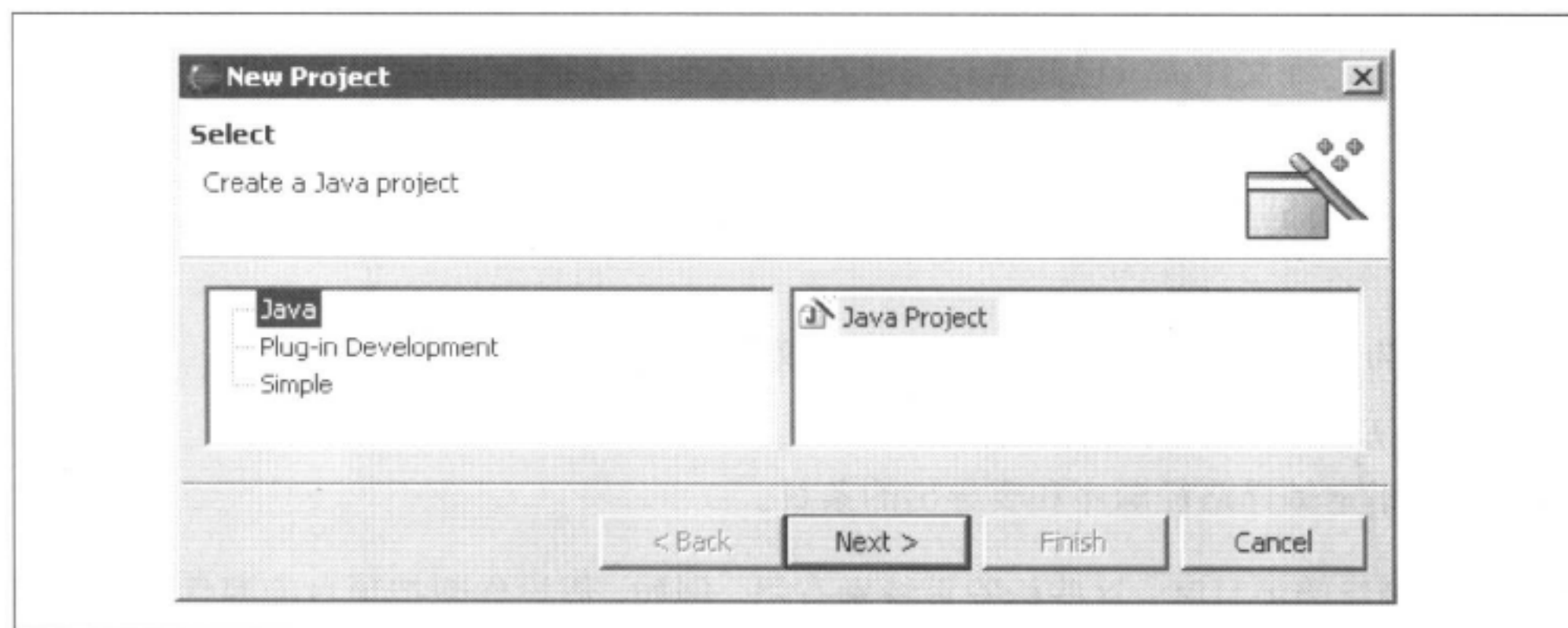


图 3-2：New Project 对话框

在下一个对话框中为项目命名，如图 3-3 所示，然后单击 Next。

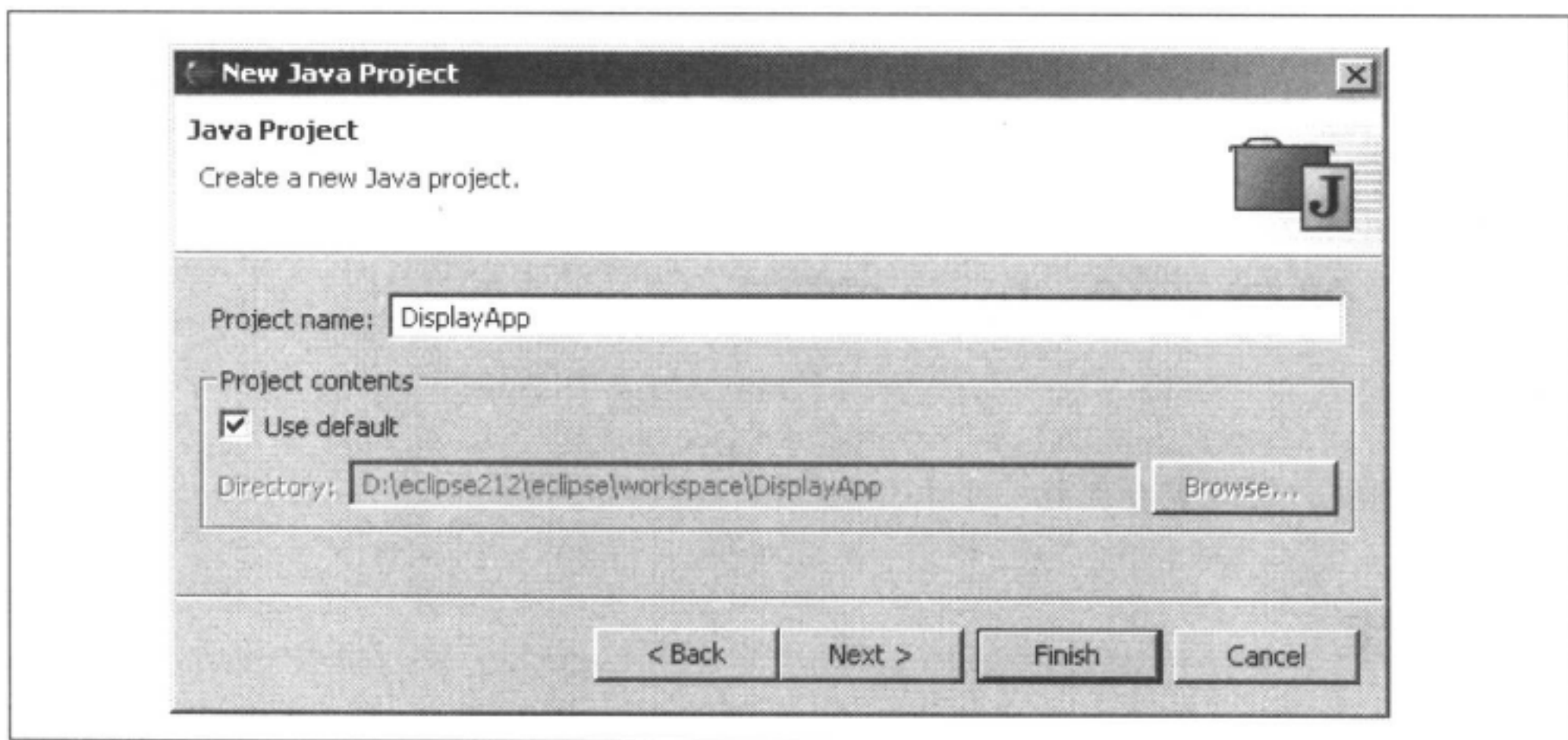


图 3-3：为项目命名

注意： 图3-3所示的Project contents选项用于指定项目的位置。把项目存储在其他软件（如Tomcat Web 服务器）能够找到的地方，是十分有用的。若想为项目选择一个自定义的位置，可取消 Use default 复选框，然后浏览到一个新的目录。

这一系列操作的最后一个对话框如图 3-4 所示，它包含 4 个标签页；下面是这些标签页的功能：

Source

用于指定源文件夹（已简单介绍过）。

Projects

用于在编译路径上指定其他项目。

Libraries

用于为编译路径指定 JAR 文件和其他文件夹。

Order and Export

用于指定编译路径顺序和要导出的条目。

在配置所创建的项目时，这些标签页特别有用。例如，假设你想把项目的源代码存储在一个名为 *src* 的文件夹中。为此，可以单击 Source 标签，单击 Add Folder，单击 Create New Folder，输入文件夹名 *src*，然后单击两次 OK。Eclipse 将询问你是否使用这个新建的文件夹作为源代码文件夹，而不是使用默认的项目文件夹，是否使用名为 *bin* 的输出版本文件夹，如图 3-5 所示。如果你想把 *src* 文件夹设置为项目的源代码文件夹，而且想使用 *bin* 文件夹来存储编译后的输出，则单击 Yes。

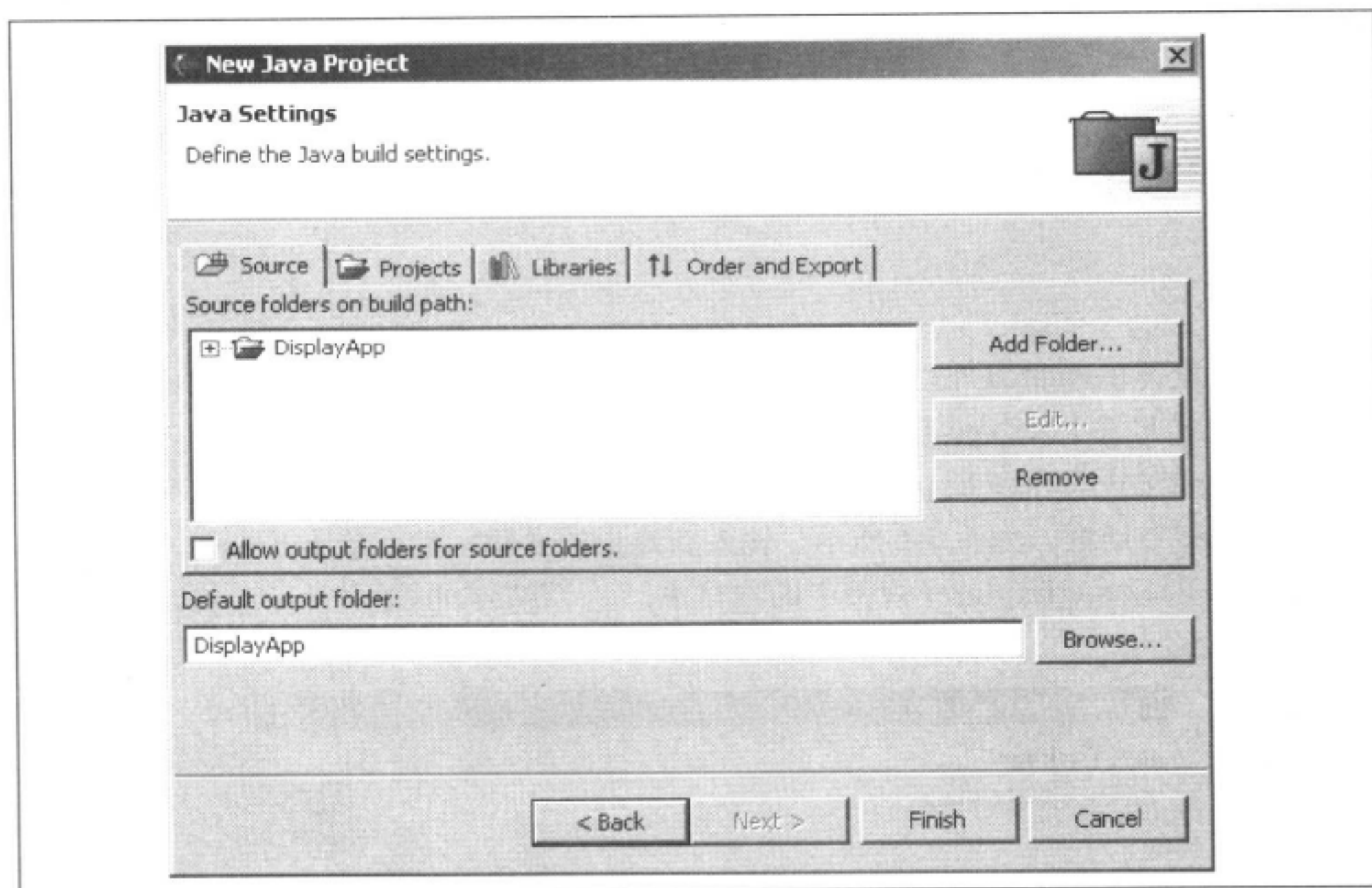


图 3-4：设置项目选项

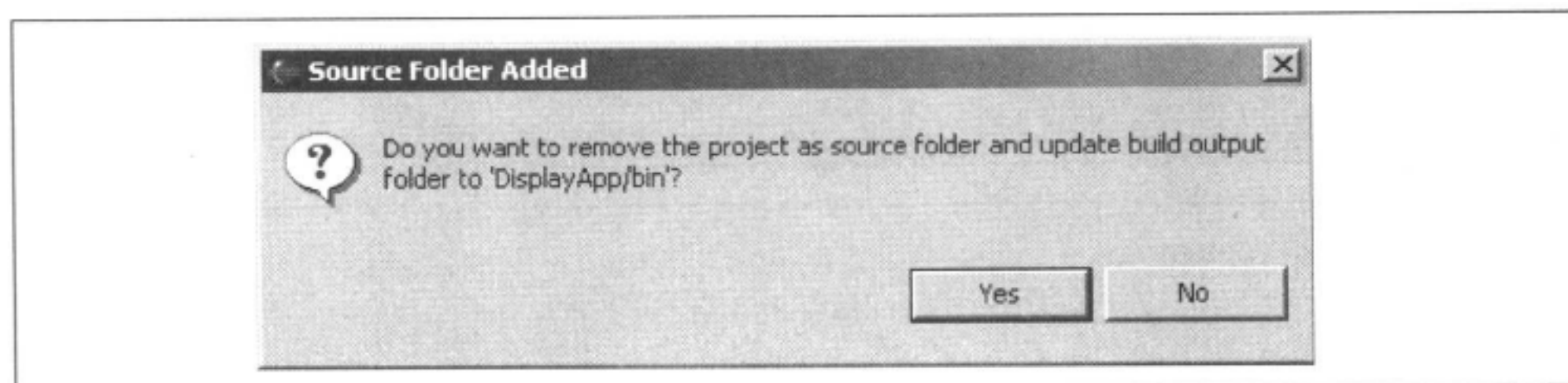


图 3-5：创建一个输出文件夹

参考

Eclipse (O'Reilly) 一书的第 2 章。

3.3 创建 Java 包

问题

你想把代码输出到一个 Java 包中。

解决方案

选择 File → New → Package，或者在 Java 透视图中右击 Package Explorer 视图，并选择 New → Package。另外，可以在创建新类时指定包。

讨论

除了最无足轻重的项目以外，在所有项目中，通常都把 Java 代码放入一个包中。要创建一个新的包，可在 Package Explorer 视图中选择一个项目，然后选择 File → New → Package，或者在 Package Explorer 视图中右击，并选择 New → Package。这将打开 New Java Package 对话框，如图 3-6 所示。输入新建包的名称，然后单击 Finish。新建的包将被添加到 Package Explorer 视图中的项目中。

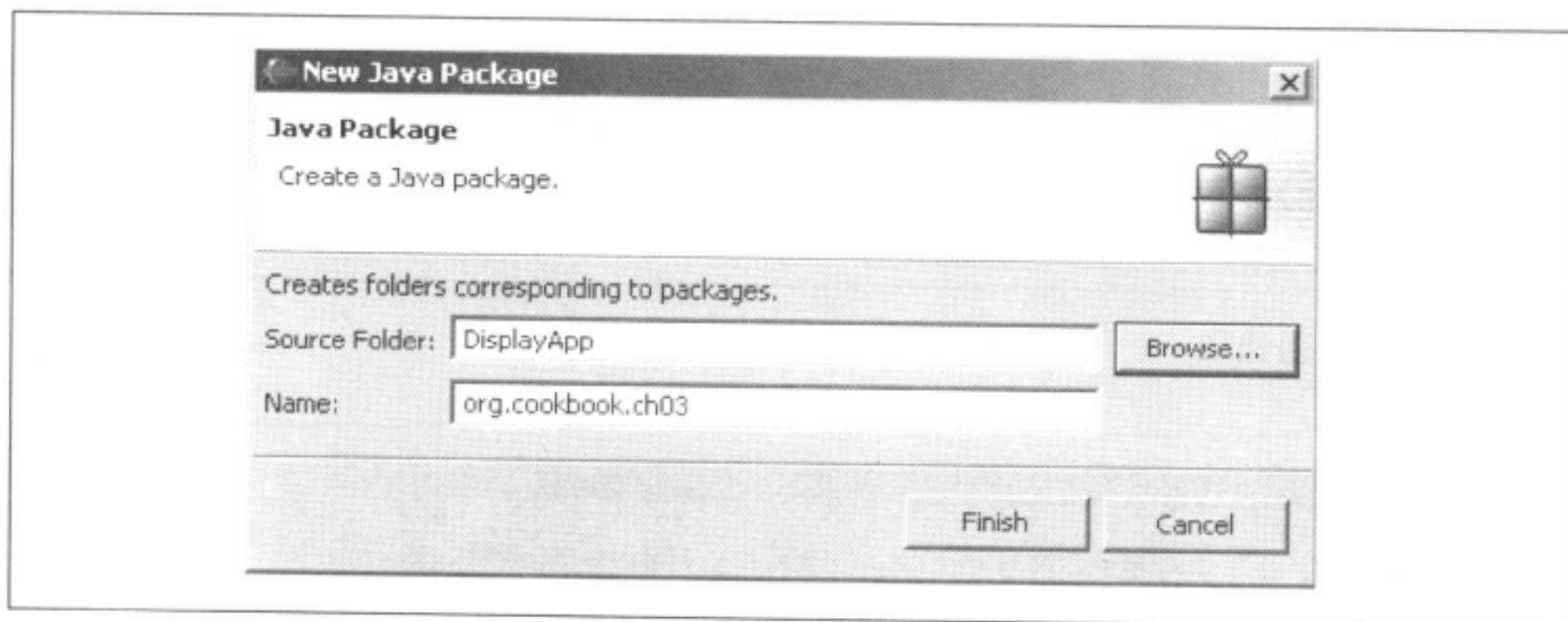


图 3-6：创建一个新的包

新建的包出现在项目中时，通过右击该包，然后选择 New → Class，可以向包中添加类。

注意：虽然可以以这种方式在 Eclipse 中创建包，但通常应该在创建类时创建包（除非你是特意地将项目分解为多个包）。3.4 节将介绍创建类的同时创建包的过程。

在访问另一个包中的类时，必须使用类的完全限定名称将类导入代码：

```
import org.cookbook.ch03.DisplayApp;

public class DisplayApp {

    public static void main(String[] args){
        DisplayApp display = new DisplayApp();
        display.print();
    }
}
```



```
    }  
}
```

注意： 你想使用的一些代码可能全部在另一个项目中。在创建一个项目时，可以指定需要在编译路径中包括的其他项目（参阅3.4节）。此外，还可以通过选择Project → Properties → Java Build Path，在现有项目中指定编译路径中应包含的其他项目。

3.4 创建 Java 类

问题

你想新建一个 Java 类，并在其中输入代码。

解决方案

选择一个项目，然后选择 File → New → Class，或者在 Package Explorer 视图中右击一个项目，并选择 New → Class。

讨论

当选择 File → New → Class，或者在 Package Explorer 视图中右击一个项目，并选择 New → Class 时，将打开 New Java Class 对话框，如图 3-7 所示。

注意该对话框中的选项。可以指定包含类的包的名称（如果该包不存在，将创建它）。可以设置类的访问限定符，还可以设置类的超类，从而实现继承。还可以指定类要实现的接口以及要创建的一组方法的占位程序：main 方法、构造函数以及继承的抽象方法。

注意： 如果要将一个类包含在另一个类中，可以在 New Java Class 对话框的 Enclosing type 文本框中输入包含类的名称。如果在 Package Explorer 视图中右击包含类，并选择 New → Class，当该对话框打开时包含类的名称将出现在 Enclosing type 文本框中；然而，除非选中了该文本框旁边的复选框，否则不会使用该包含类。

创建一个匿名的内部类

在 Eclipse 3.0 中，代码助手（也叫做内容助手）可以帮助你创建匿名的内部类。把光标定位在类实例创建的开花括号之后，然后按组合键 Ctrl-Space（或者选择 Edit → Content Assist）。代码助手可自动创建匿名内部类的主体，包括需要实现的所有方法。棒极了。关于代码助手的详细内容，请参阅第 2 章。

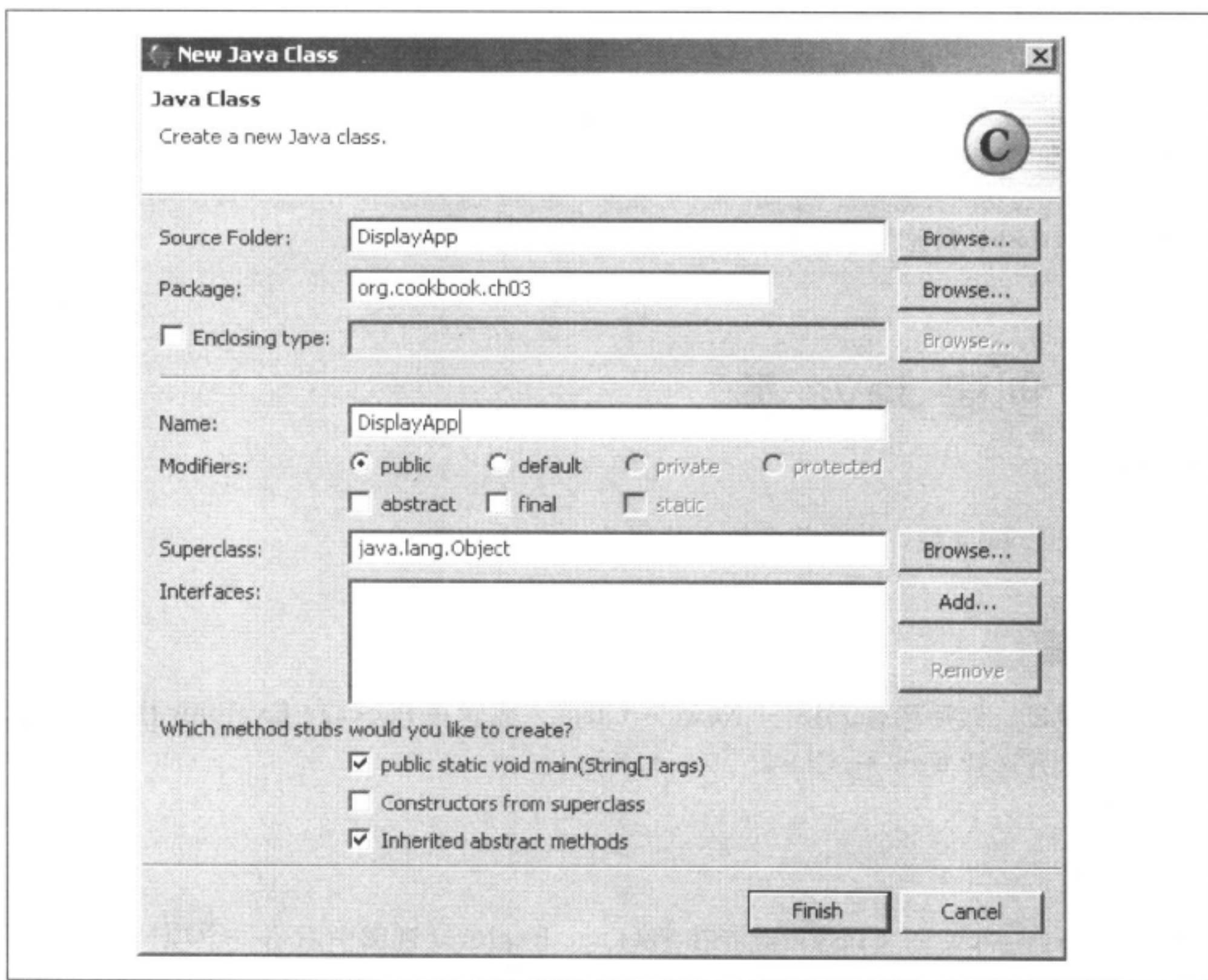


图 3-7：新建一个 Java 类

3.5 创建 Java 方法

问题

你想创建一个 Java 方法，以运行代码。

解决方案

自己在 JDT 编辑器中输入代码，或者让代码助手帮助你完成。

讨论

Java 代码必须放在方法中才能运行。Eclipse 使用代码助手来帮助完成这一任务。假设你想创建例 3-1 所示的代码，其中包括一个名为 `display` 的新方法。

例 3-1: DisplayApp.java 示例

```
public class DisplayApp
{
    public static void main(String[] args)
    {
        display();
    }

    private static void display()
    {
        System.out.println("No problem.");
    }
}
```

要弄清代码助手能够做什么,我们创建了 *DisplayApp* 项目,并让 Eclipse 生成一个 main 方法:

```
public class DisplayApp {

    public static void main(String[] args) {

    }

}
```

不必以手工方式在 *display* 方法中输入代码,使用代码助手即可。把光标移动到 main 方法的下方,并输入关键字 *private*,使 *display* 成为一个 *private* 方法。然后,按组合键 *Ctrl-Space*,打开代码助手,如图 3-8 所示。

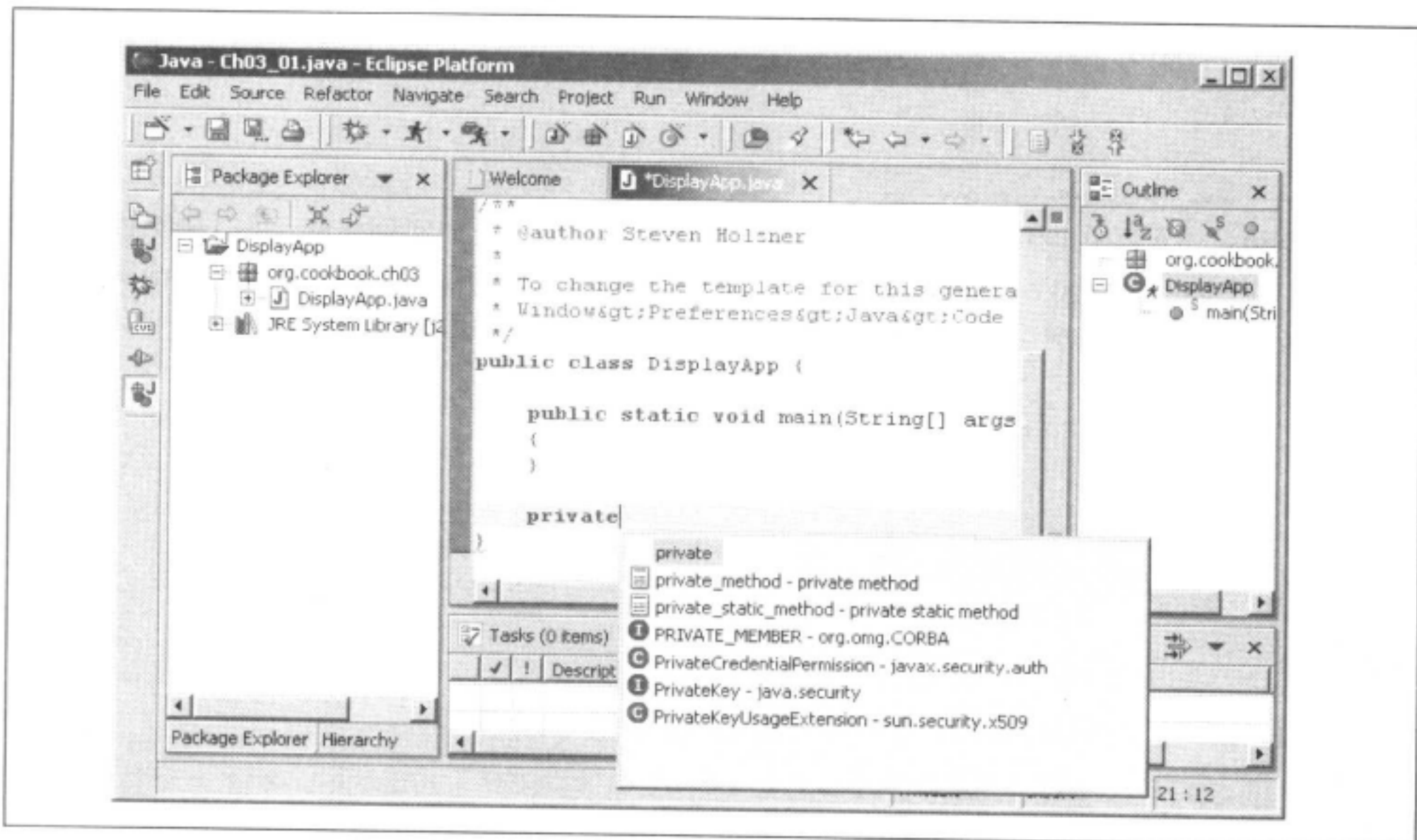


图 3-8: 新建一个方法

选择 private static 方法项，代码助手将创建方法模板，如图 3-9 所示。

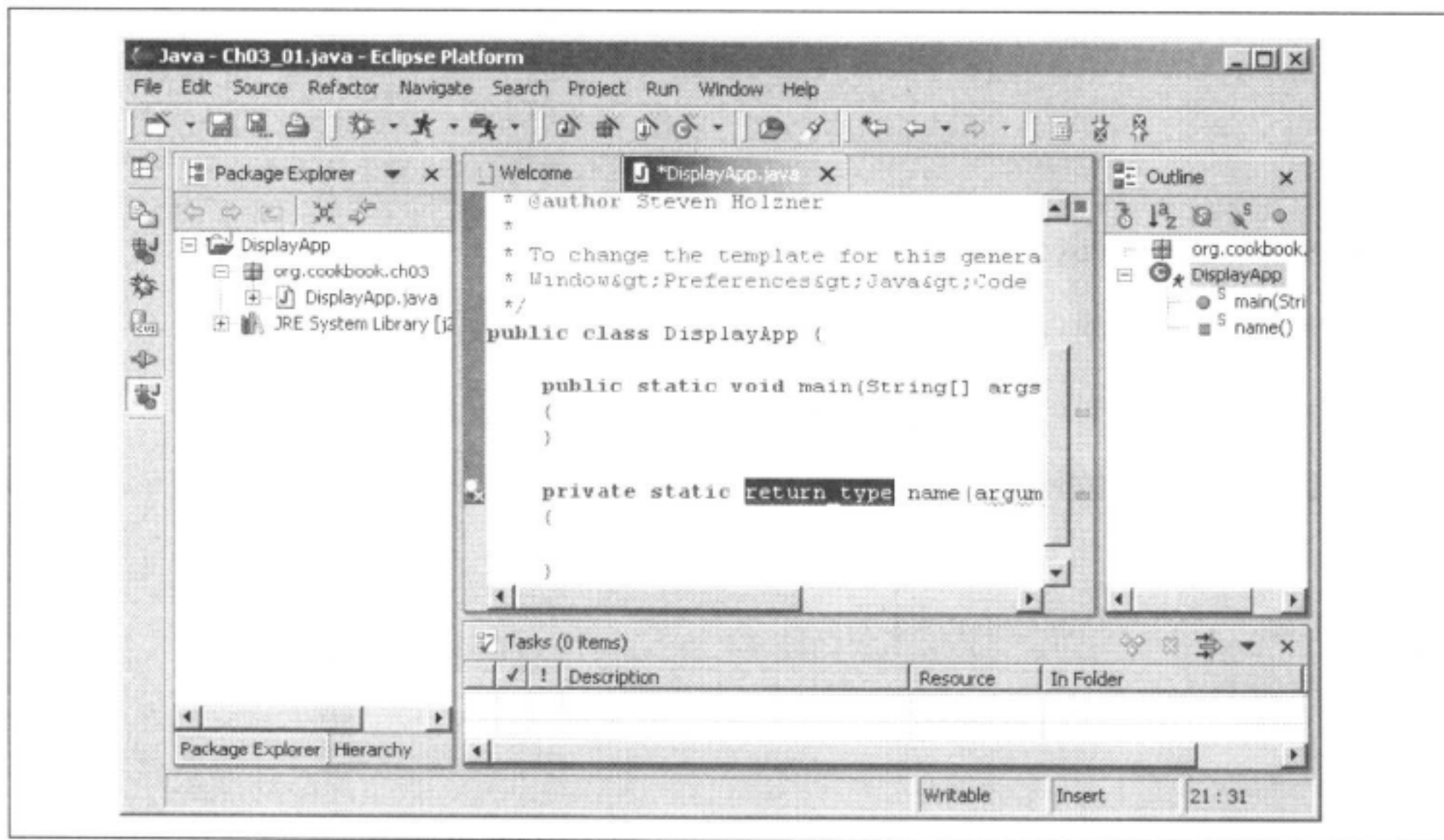


图 3-9：一个新建的方法模板

填写 return_type、name 和 arguments 项，即可得到例 3-1 所示的代码；保存文件；然后选择 Run → Run As → Java Application，运行项目。Console 视图将显示文字 "No problem."。

注意： Eclipse 还可以在调用方法时自动创建方法。为了弄清它是如何创建的，可以删除或注释掉代码中的 display 方法。单击 main 方法中 display 调用旁的 Quick Fix 灯泡图标，然后选择 Create method 'display(...)'. Eclipse 将使用要调用的方法的签名，来创建方法的框架。

3.6 覆盖 Java 方法

问题

你想覆盖一个类的方法，但首先需要知道哪些方法可以覆盖。

解决方案

选择 Source → Override/Implement Methods。

讨论

要弄清哪些方法可以覆盖，可打开一个类的派生类，然后选择 **Source** → **Override/Implement Methods**，打开 **Override/Implement Methods** 对话框，如图 3-10 所示。选择一些方法并单击 **OK**，将使 Eclipse 自动创建这些方法的占位程序。

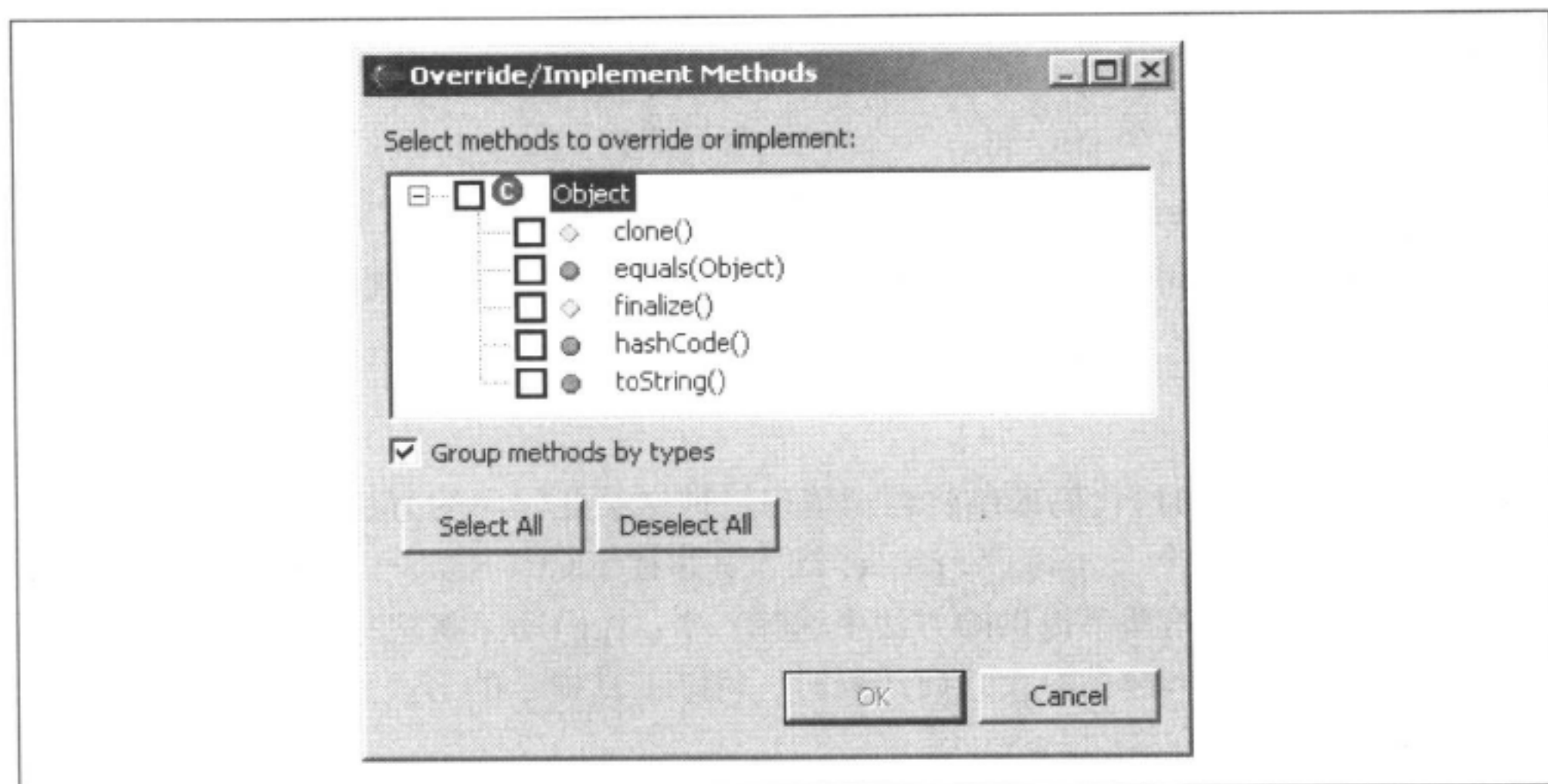


图 3-10：覆盖方法

3.7 获取方法参数的提示

问题

你想调用一个方法，但却忘记了它的参数是什么，而且你想得到一些提示。

解决方案

把光标定位到方法的参数列表中，然后按组合键 **Ctrl-Shift-Space**，或选择 **Edit** → **Parameter Hints**。

讨论

当把光标定位到一个方法的参数列表中时，可以看到一个参数提示列表。在 JDT 编辑器中，按组合键 **Ctrl-Shift-Space**，或选择 **Edit** → **Parameter Hints**，将显示一个工具提示，说明方法的参数。

3.8 插入方法参数名

问题

你忘记了要调用的一个方法的参数,而且不想查看工具提示(见上一节),你想让Eclipse在代码中填写参数名称。

解决方案

选择 Window → Preferences → Java → Editor → Code Assist, 然后选中 Fill argument names on method completion 和 Guess filled argument names 复选框。

讨论

当选中这两个复选框时,代码助手将自动填写已传递给代码中的方法的参数名称。例如,假设你已经输入了 `java.util.Arrays.fill`, 并按下 `Ctrl-Space` 组合键以调用代码助手,那么就可以从代码助手提供的方法中选择一个。代码助手将把每一个参数的名称插入代码,而当你用 `Tab` 键在参数之间切换时,将以工具提示的形式显示参数的类型,如图 3-11 所示。

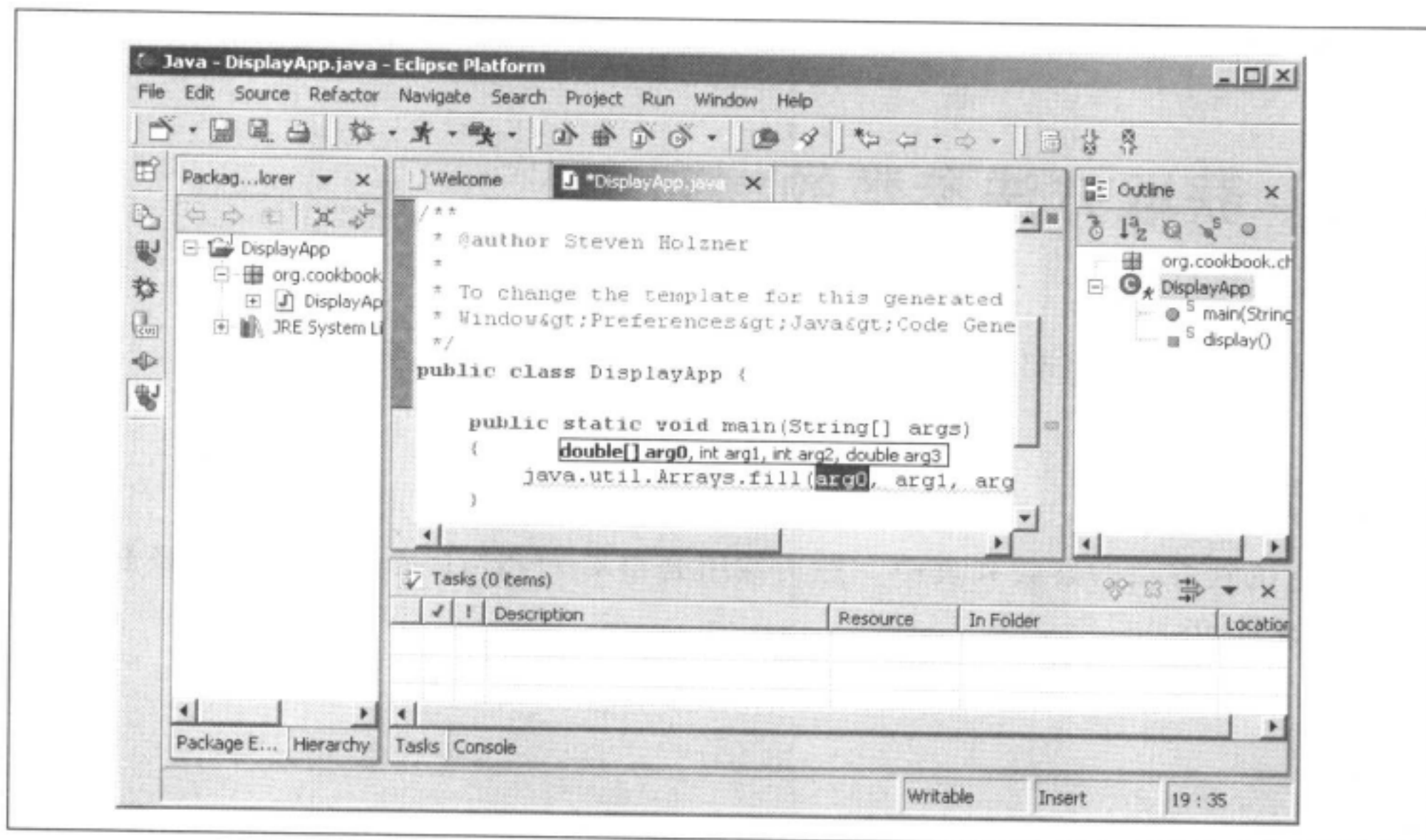


图 3-11: 自动列举参数名称

3.9 创建 getter/setter 方法

问题

你需要创建字段的 getter/setter 方法（例如，当使用 JavaBeans 创建属性时），而且想找一种快捷的方式。

解决方案

选择 Source → Generate Getter and Setter。

讨论

假设代码中使用一个名为 `text` 的字段来存储要显示的文本：

```
public class DisplayApp {  
  
    static String text = "No problem.";  
  
    public static void main(String[] args)  
    {  
        System.out.println(text);  
    }  
}
```

可以不把数据存储在一个简单的字段中，而是创建该数据的 getter 方法和 setter 方法，使得从类的外部访问数据更加安全。要自动创建 getter 方法和 setter 方法，可选择 Source → Generate Getter and Setter，打开如图 3-12 所示的对话框。选择需要重建 getter 方法和 setter 方法的字段以及要创建的方法，然后单击 OK。

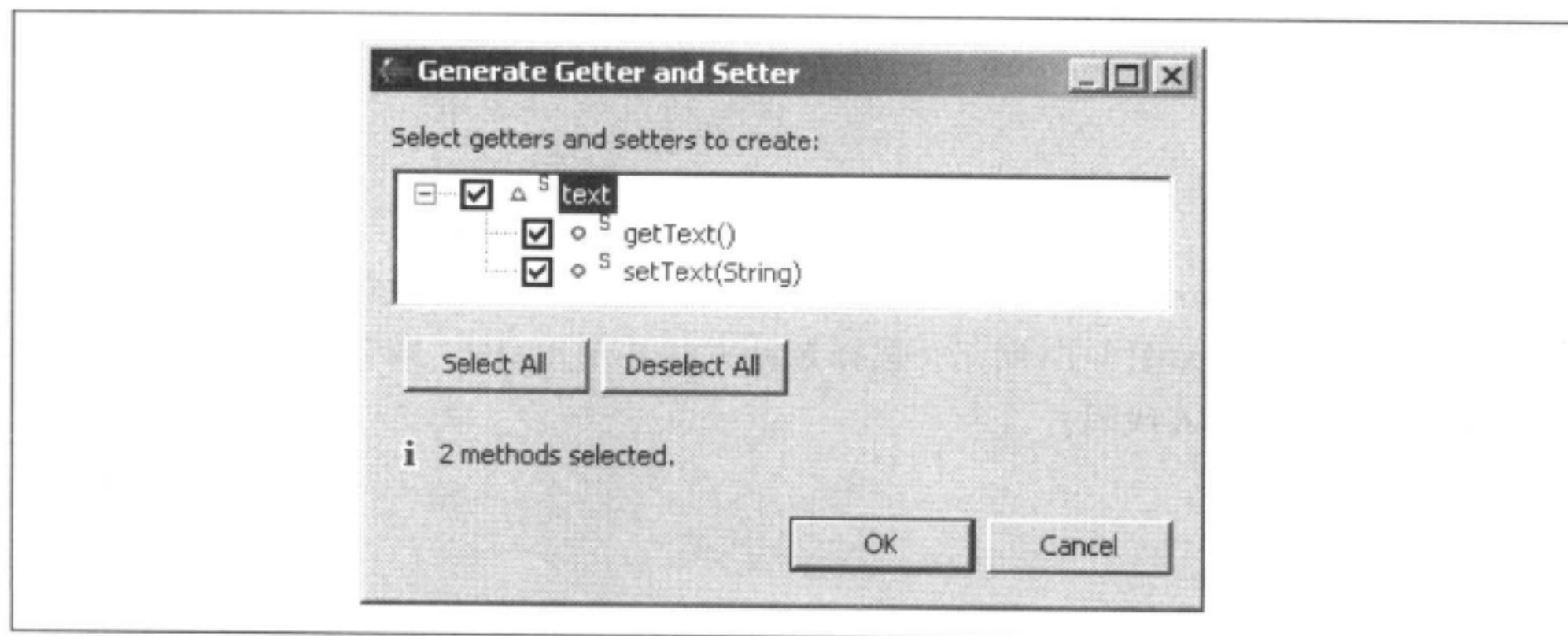


图 3-12：创建 getter/setter 方法

这将新建如下所示的 getter 方法和 setter 方法：

```
public class DisplayApp {  
  
    static String text = "No problem.";  
  
    public static void main(String[] args)  
    {  
        System.out.println(getText());  
    }  
    /**  
     * @return  
     */  
    public static String getText() {  
        return text;  
    }  
  
    /**  
     * @param string  
     */  
    public static void setText(String string) {  
        text = string;  
    }  
}
```

3.10 创建委托方法

问题

你想为字段创建一个委托（delegate）方法。

解决方案

选中字段的声明，然后选择 Source → Generate Delegate Methods。

讨论

当选择 Source → Generate Delegate Methods 时，将显示一个方法列表，字段将为这些方法创建委托方法，如图 3-13 所示。选择各种方法并单击 OK，即可为这些方法创建委托方法，同时将其插入代码。

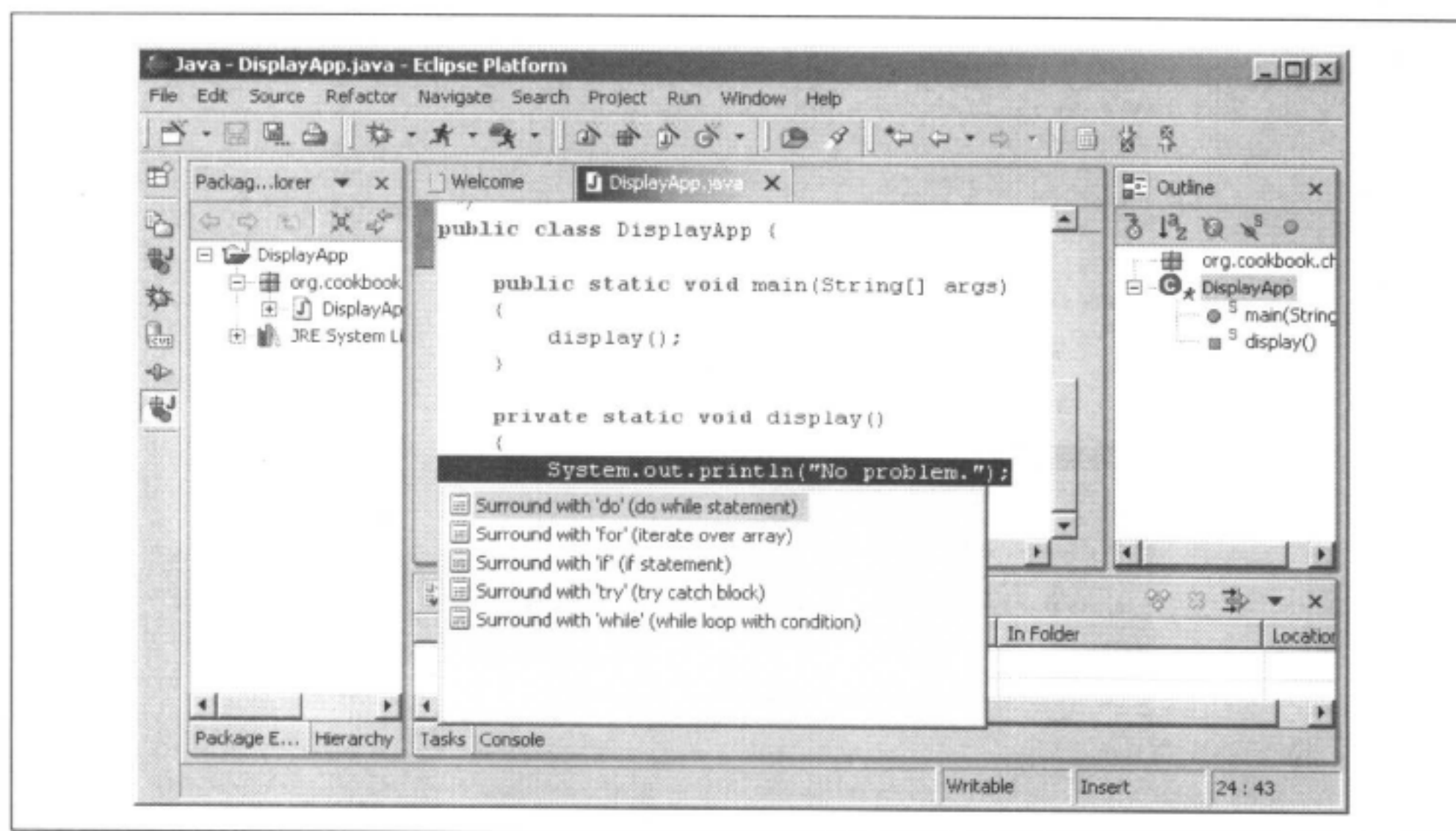


图 3-13：创建委托方法

3.11 用 do/for/if/try/while 块包围代码

问题

你想把一段代码装入 `try/catch` 块或 `do/for/if/while` 结构中。

解决方案

选中要包围的代码行，按组合键 `Ctrl-1`（或选择 `Edit → Quick Fix`），以列出所有的选项，然后从中选择一种。

讨论

图 3-14 显示了当选择了代码并按下组合键 `Ctrl-1` 时显示的选项列表。选择其中一个选项，即可用所选的结构包围选中的代码。

注意：这样做的好处之一是，如果用 `try/catch` 块包围代码，Eclipse 将自动推测所有可能的异常，并捕获异常。

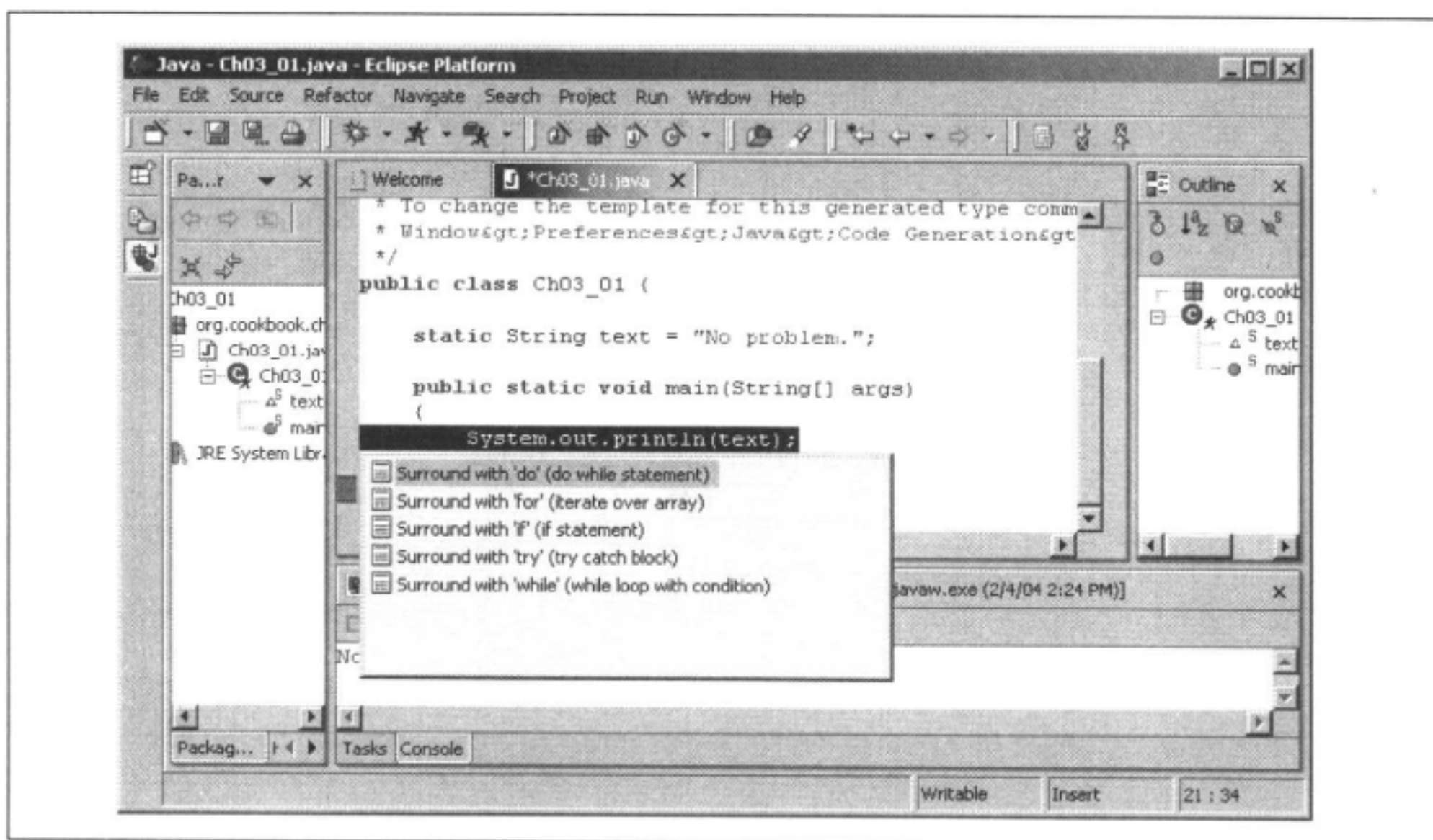


图 3-14：将代码装入块中

参考

Eclipse (O'Reilly) 一书的第 2 章。

3.12 查找匹配的花括号

问题

代码越来越长，而且很难找到与开花括号匹配的闭花括号，反之亦然。

解决方案

选择开花括号或闭花括号，并按组合键 Ctrl-Shift-P（或选择 Navigate → Go To → Matching Bracket）。

讨论

对于长代码清单而言，这是一个有用的窍门。要将一个开花括号与相应的闭花括号相匹配，只需选择开花括号，并按下组合键 Ctrl-Shift-P 即可。Eclipse 将加亮显示包含闭花括号的代码行。

注意： 也可以在开花括号之前或闭花括号之后双击，这将自动选中两个花括号之间的文本，从而快速找到闭花括号。这种技术不仅适用于花括号，而且适用于圆括号。

3.13 自动包围字符串

问题

文本字符串太长，超过了屏幕的宽度，要滚动到字符串结尾才能进行编辑，而你已经对此感到厌倦。

解决方案

让Eclipse来分隔字符串。只需将光标定位到要分隔的引用文本中，然后按Enter键即可。

讨论

Eclipse可以令人满意地处理Java代码中的字符串分隔问题。假设有下面的文本字符串：

```
String text = "This is a long string of text.";
```

可以把光标定位到该字符串的中间，并按下Enter键，通过按如下方式分隔引用的字符串，Eclipse对Java代码进行了正确的处理：

```
String text = "This is a long " +  
    "string of text.";
```

注意： 可以在Typing页面自定义这种分隔方式。要访问Typing页面，可选择Window → Preferences → Java → Editor → Typing。

3.14 创建构造函数

问题

你想让Eclipse在类中添加一个构造函数，包括对超类的构造函数的调用。

解决方案

选择Source → Add Constructor from Superclass。

讨论

例如，如果有下面的代码：

```
public class DisplayApp {  
    static String text = "No problem.";  
    public static void main(String[] args)  
    {  
        System.out.println(text);  
    }  
}
```

并且你选择了 Source → Add Constructor from Superclass，那么 Eclipse 将提供下面的代码：

```
public class DisplayApp {  
    static String text = "No problem.";  
    /**  
     *  
     */  
    public DisplayApp() {  
        super();  
        // TODO Auto-generated constructor stub  
    }  
    public static void main(String[] args)  
    {  
        System.out.println(text);  
    }  
}
```

注意： 也可以在创建类时自动创建构造函数。

Eclipse 3.0

在 Eclipse 3.0 中，可以创建一个或多个字段赋值的构造函数。选择 Source → Generate Constructor using Fields，打开如图 3-15 所示的对话框。

选择两个 String 字段，text 和 message，创建下面的构造函数：

```
public DisplayApp(String text, String message) {  
    super();  
    this.text = text;  
    this.message = message;  
}
```

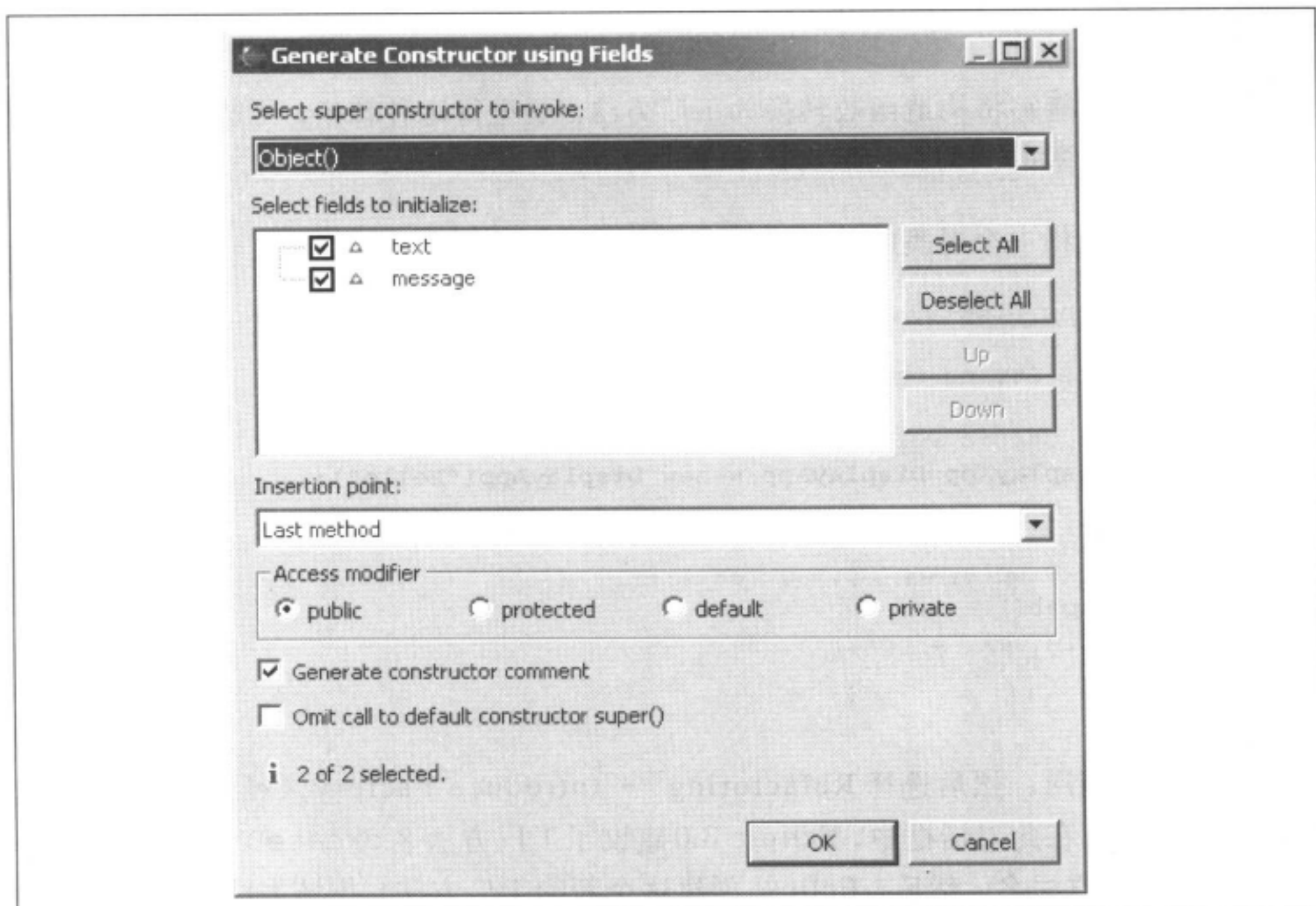


图 3-15：创建为字段赋值的构造函数

参考

Eclipse (O'Reilly) 一书的第 2 章。

3.15 将构造函数转换为工厂方法

问题

你想将一个构造函数转换为工厂方法

解决方案

在 Eclipse 3.0 中，在 JDT 编辑器中选择一个构造函数声明或对构造函数的调用，然后选择 Refactoring → Introduce Factory。

注意： 这是一个仅适用于 Eclipse 3.0 的解决方案。在 3.0 之前的 Eclipse 版本中，没有相应的解决方案。

讨论

Eclipse 3.0是你能够将构造函数转换为工厂方法。要进行这种转换,可选择一个函数声明或对该构造函数的调用,然后选择 Refactoring → Introduce Factory。

例如,假设有这样一个对类的构造函数的调用:

```
public class DisplayApp {  
    private String text;  
  
    public static void main(String[] args) {  
        DisplayApp DisplayApp = new DisplayApp("Hello");  
    }  
  
    public DisplayApp(String text) {  
        super();  
        this.text = text;  
    }  
}
```

选择构造函数调用,然后选择 Refactoring → Introduce Factory,可打开 Introduce Factory对话框。在此对话框中,Eclipse 3.0建议了工厂方法名 createDisplayApp。单击 OK,接受该方法名。然后,Eclipse 创建这个新的工厂方法,用该方法的调用替换所有构造函数调用,使原来的构造函数变为私有的:

```
public class DisplayApp {  
    private String text;  
  
    public static void main(String[] args) {  
        DisplayApp DisplayApp = createDisplayApp("Hello");  
    }  
  
    public static DisplayApp createDisplayApp(java.lang.String text) {  
        return new DisplayApp(text);  
    }  
  
    /**  
     * @param text  
     */  
    private DisplayApp(String text) {  
        super();  
        this.text = text;  
    }  
}
```

3.16 注释掉一段代码

问题

你想注释掉一长段代码。

解决方案

选择 Source → Comment。

讨论

你需要做的全部工作就是选择要注释掉的代码行，然后选择 Source → Comment。一个单行注释符//将出现在所有代码行的前面。要取消注释，只需选择Source → Uncomment即可。

Eclipse 3.0

在Eclipse 3.0中，还可以使用块注释符(/*...*/)注释代码。只需选择Source → Add Block Comment即可。要删除注释符，选择Source → Remove Block Comment即可。

参考

Eclipse (O'Reilly) 一书的第2章。

3.17 创建工作集

问题

Package Explorer视图充满了各种项目。

解决方案

创建一个工作集，然后在Package Explorer视图中，只选择该工作集中的项目。

讨论

在第1章中，你已经了解了如何删除项目而不删除项目的内容，使你在以后能够将项目

导入到 Package Explorer 视图中。也可以通过创建工作集——在 Package Explorer 视图中作为一组进行处理的项目的集合——来定义 Package Explorer。

例如，假设有 4 个项目，*DisplayApp*、*DisplayApp2*、*DisplayApp3* 和 *DisplayApp4*，如图 3-16 所示。然而，你只需要使用两个项目，*DisplayApp* 和 *DisplayApp2*。

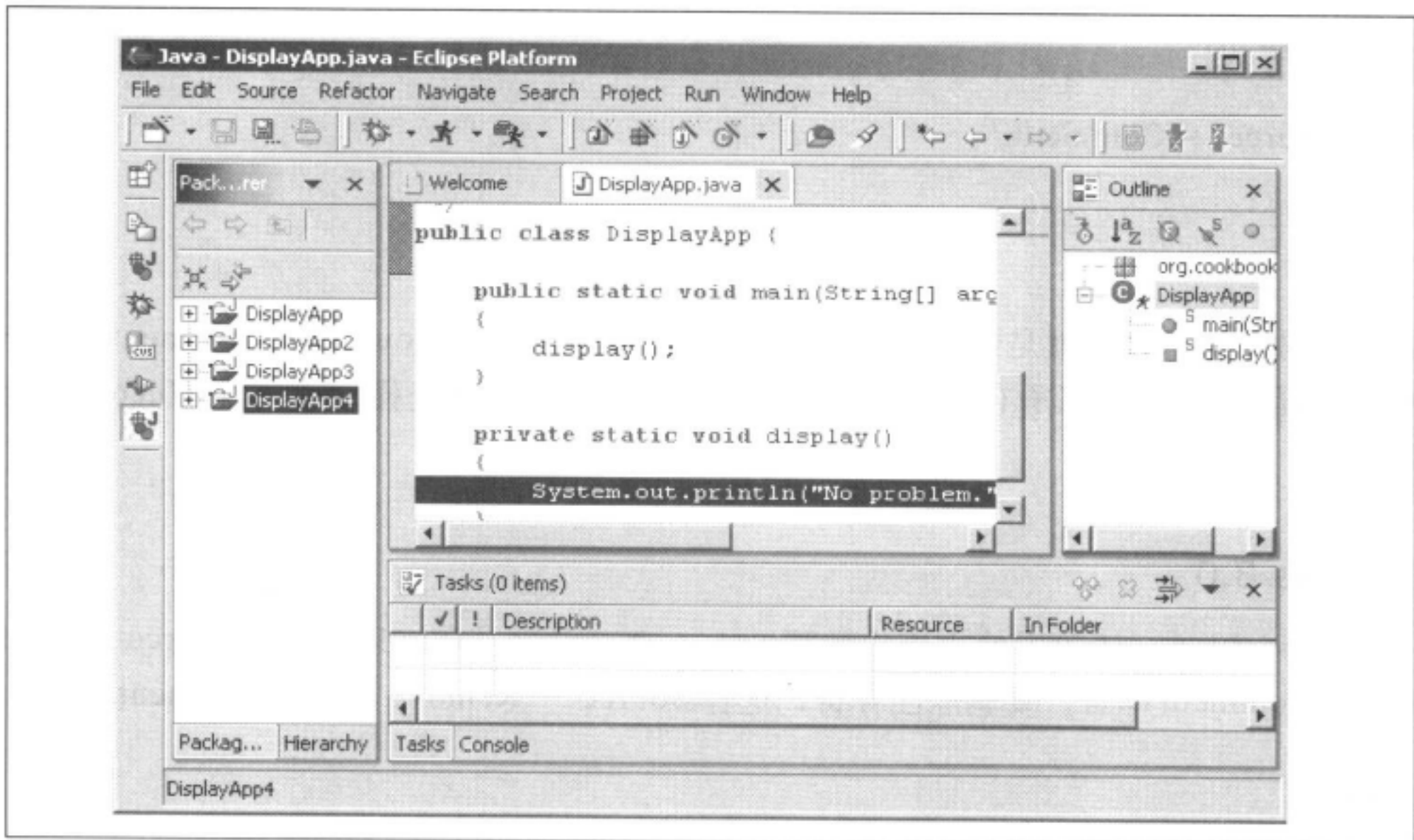


图 3-16: Package Explorer 视图中的 4 个项目

要创建只包含这两个项目的工作集，可单击 Package Explorer 视图的下拉菜单（视图顶部的倒黑三角型），并选择 Select Working Set，打开 Select Working Set 对话框。要新建一个工作集，单击 New 按钮即可。这将打开 New Working Set 对话框。在 Working set type 框中，选择 Java，然后单击 Next 按钮。现在选择要加入该工作集的项目，如图 3-17 所示，命名工作集（在本例中，命名为 Two Projects），然后单击 Finish。

当单击 Finish 按钮时，Select Working Set 对话框重新打开，其中包含 Two Projects 工作集，如图 3-18 所示。

选择新建的工作集。并单击 OK。只有所选工作集中的两个项目出现在 Package Explorer 视图中，如图 3-19 所示。使用这种方法，可以清理 Package Explorer 视图。

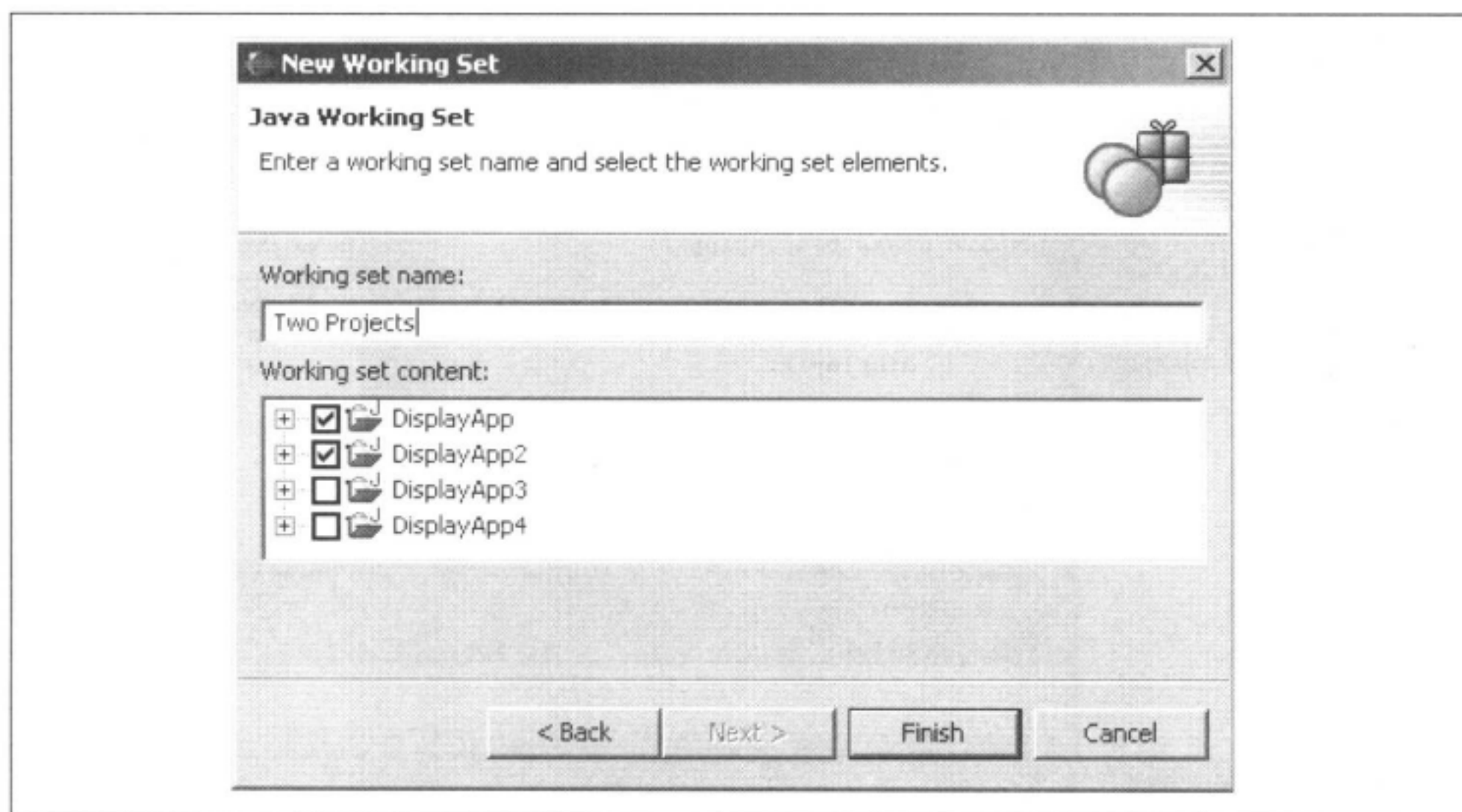


图 3-17：创建一个工作集

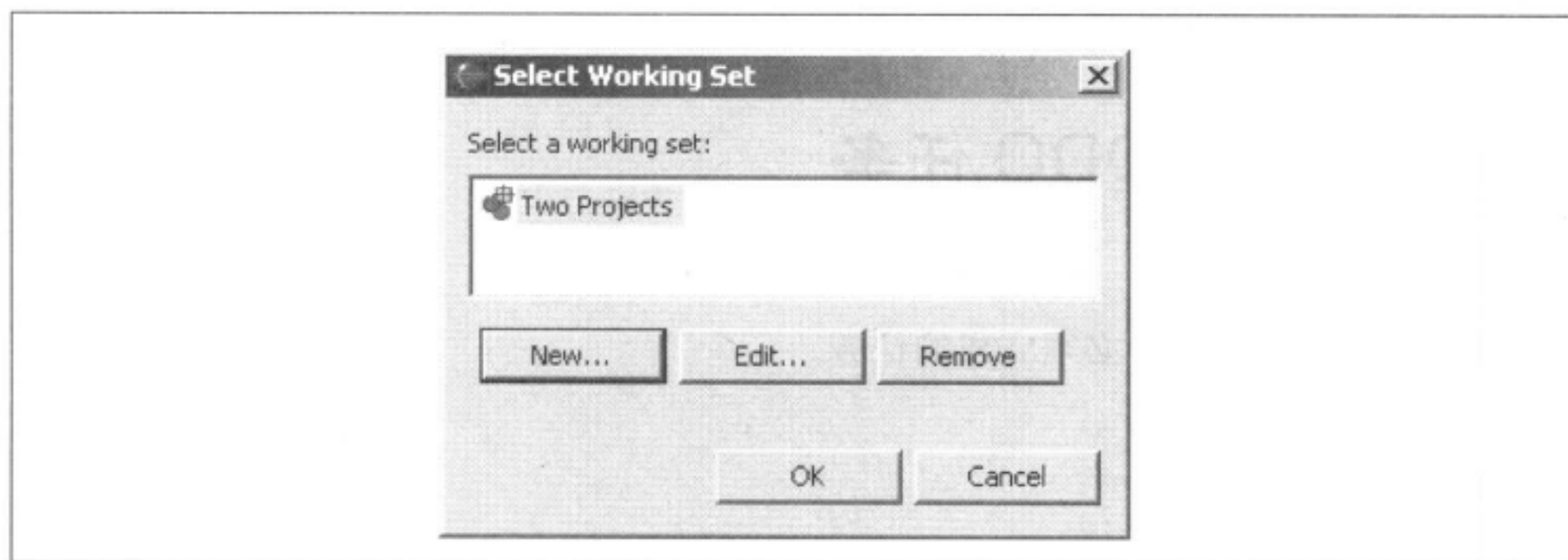


图 3-18：选择一个工作集

注意： 注意，虽然工作集决定了哪些项目出现在 Package Explorer 视图中，但在 Navigator 视图中仍然可以看到所有 4 个项目。

要取消工作集，可以从 Package Explorer 视图的下拉菜单中选择 Deselect Working Set。要编辑工作集的成员，可选择 Edit Active Working Set。

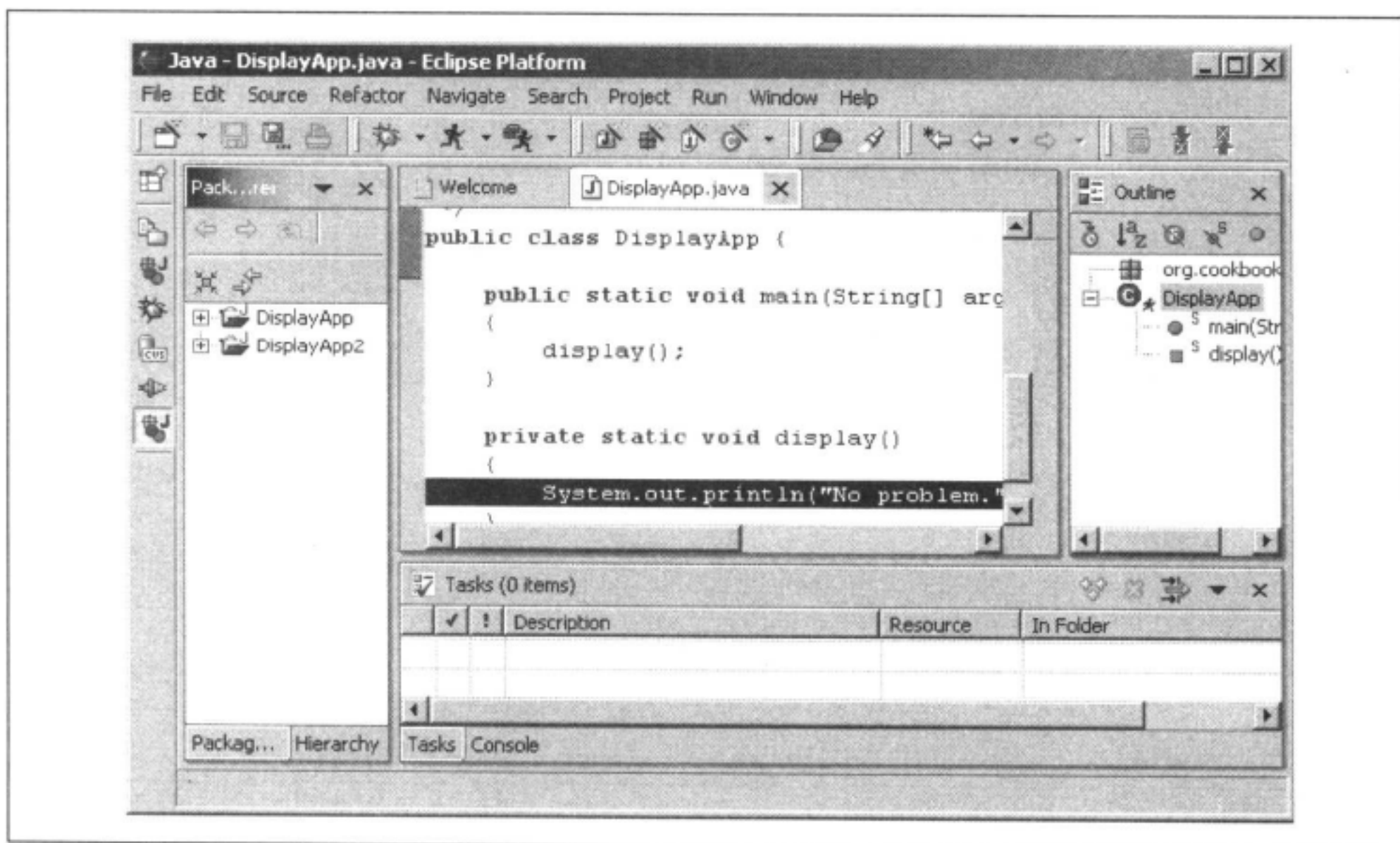


图 3-19：使用工作集

3.18 创建 TODO 任务

问题

你想使用注释跟踪代码中必须完成的任务。

解决方案

在代码中添加 TODO 注释；这些注释将出现在 Tasks 视图中。

讨论

在单行注释中添加文本 TODO，可以将该注释文本添加到 Tasks 视图中，使你能够在在一个位置跟踪所有任务。图 3-20 显示了一个示例。

Eclipse 3.0

在 Eclipse 3.0 中，Tasks 视图取代了 Problems 视图，所以，默认情况下，Tasks 视图并不出现在 Java 透视图。要显示 Tasks 视图，可选择 Window → Show View → Other → Basic → Tasks。

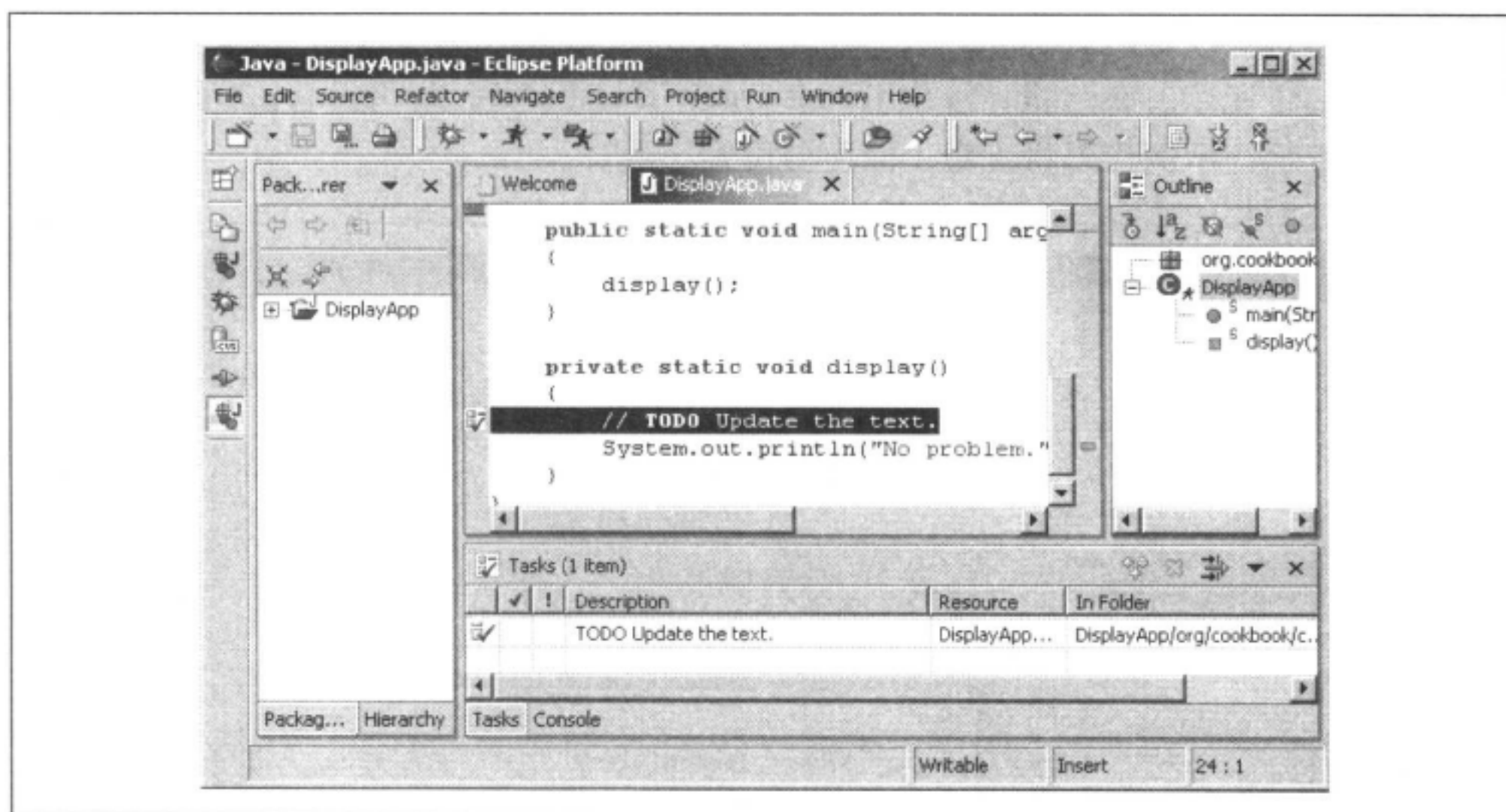


图 3-20: Tasks 视图中的 TODO 注释

3.19 自定义代码助手 问题

代码助手是一个很好的工具，但它的一些特性并不适合你的编码风格。

解决方案

通过选择 Window → Preferences → Java → Code Formatter，即可自定义代码助手。

讨论

使用代码助手时，一个常见的问题是，它把花括号与其他代码放在同一行上。

```
public void display() {  
    System.out.println("No problem.");  
}
```

然而，有些程序员喜欢花括号单独占用一行：

```
public void display()  
{  
    System.out.println("No problem.");  
}
```

通过选择 Window → Preferences → Java → Code Formatter，可以自定义花括号的位置，如图 3-21 所示。在这里，选中 Insert a new line before an opening brace 复选框，如图 3-21 所示；示例代码将改变，以便与设置一致。

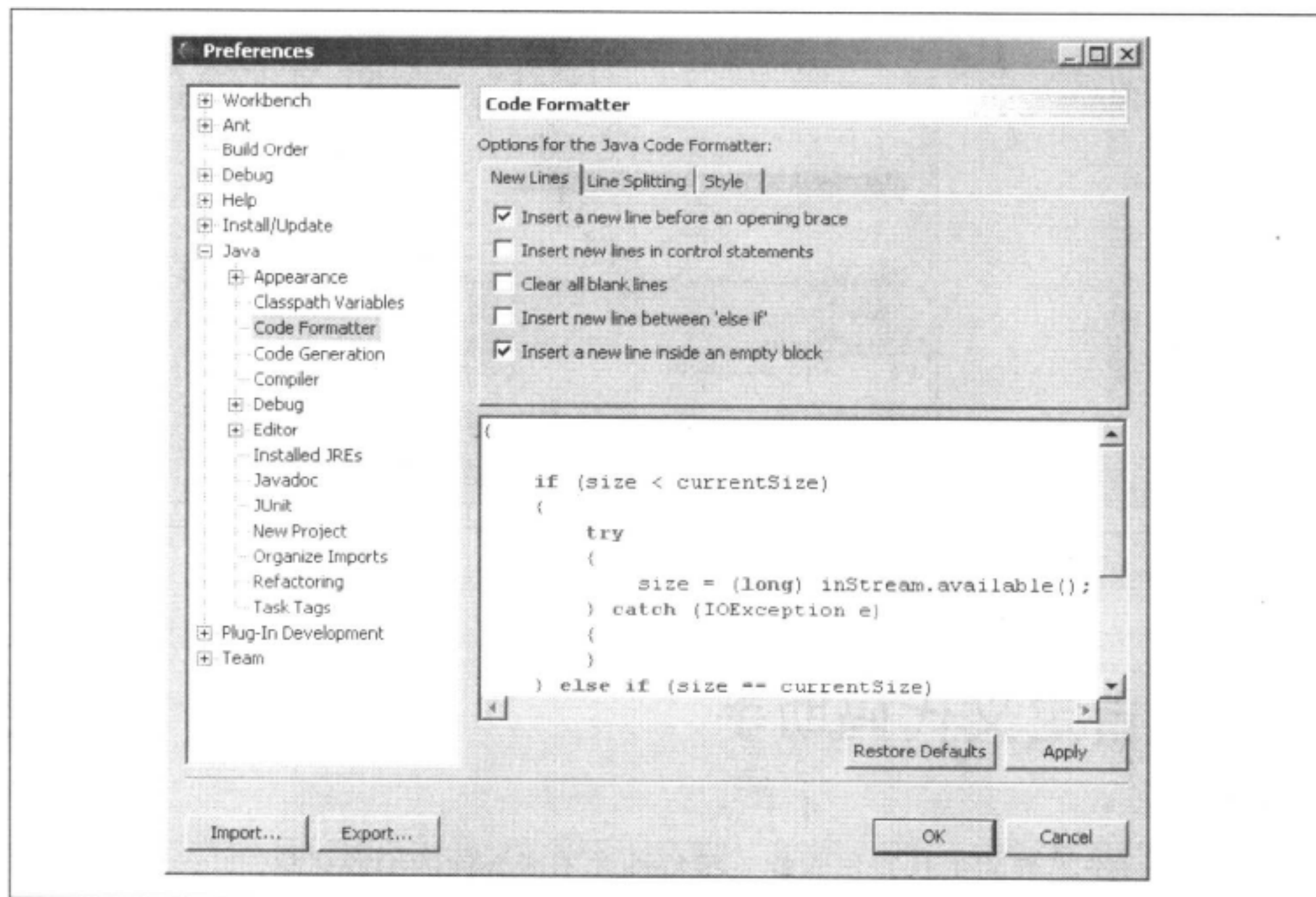


图 3-21：自定义代码助手

在 Eclipse 中，还可以创建新的代码助手项目。如果你想创建一个快捷方式以便打印当前日期，可选择 Window → Preferences → Java → Editor → Templates。在本例中，我们要新建一个名为 `date` 的快捷方式，用于打印日期，如图 3-22 所示。我们使用下面的代码：

```
System.out.println("${date}");
```

来打印日期。

注意：除了 `${date}` 以外，还可以使用其他词汇，如 `${cursor}`，表明执行插入操作后光标的位置。当在 New Template 对话框中输入 “`${`” 时，代码助手将显示在代码助手表达式中可以采用的值。

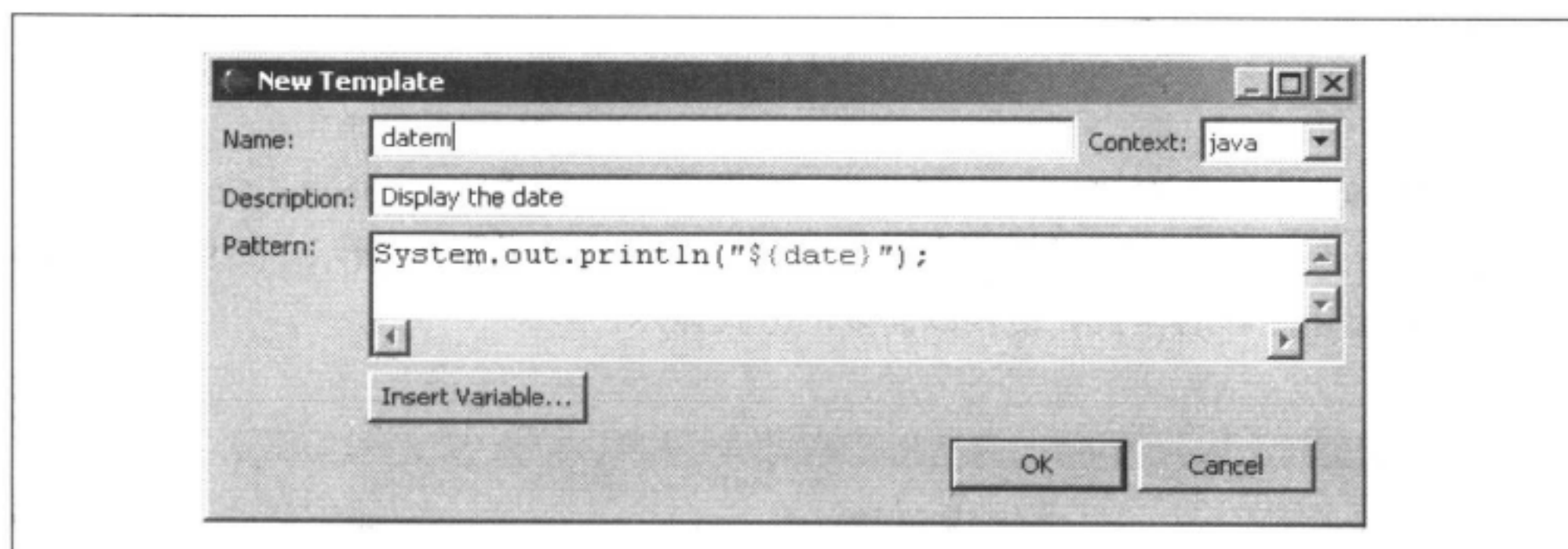


图 3-22：为代码助手新建一个快捷方式

现在，当在代码中输入 `datem` 并按组合键 `Ctrl-Space` 时，待命助手将输入打印当前日期所需的代码，如图 3-23 所示。事实上，如果输入字母 “d”，然后按组合键 `Ctrl-Space`，代码助手将列出以字母 “d” 开头的所有选项，包括 `datem`。

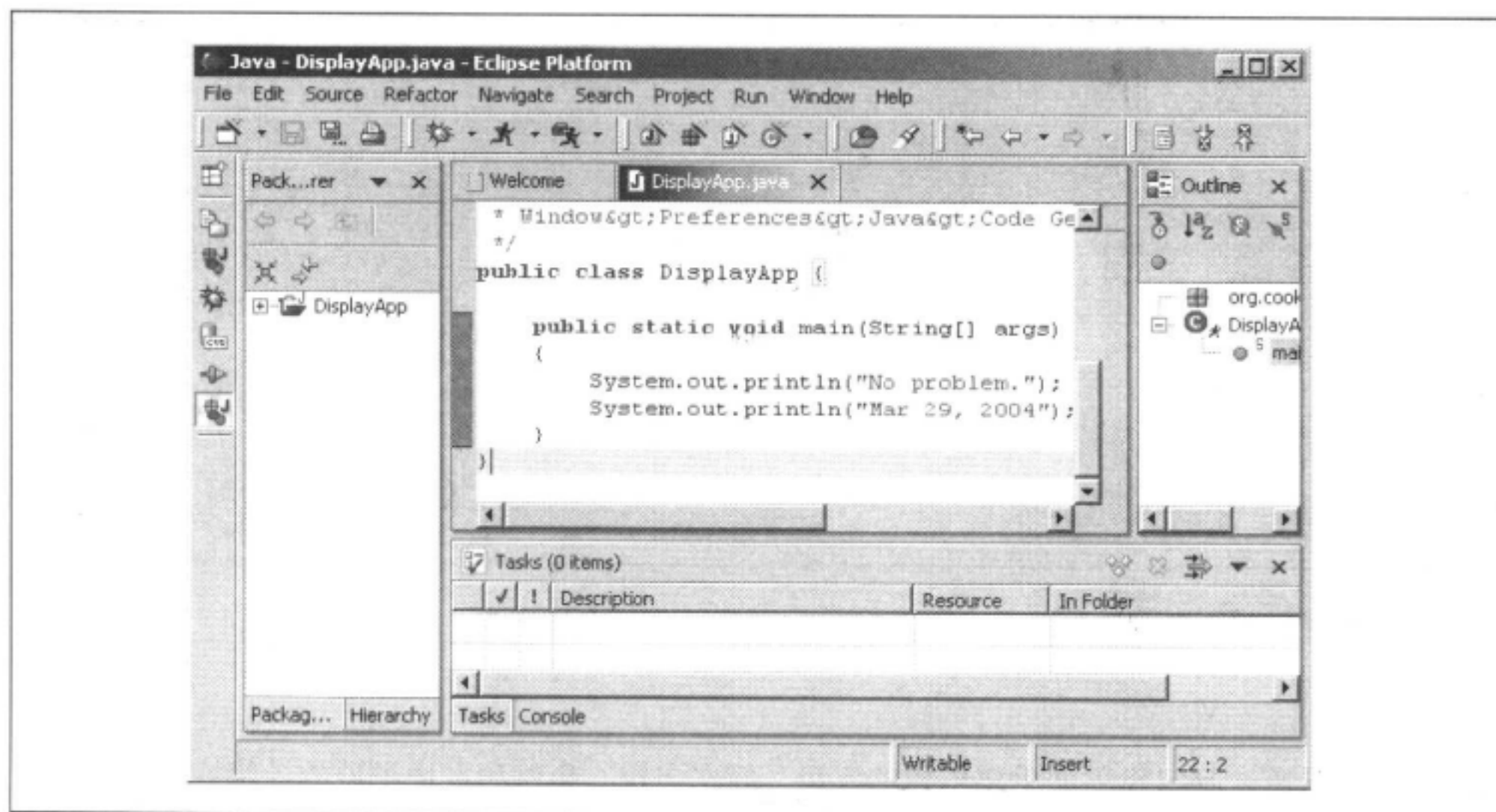


图 3-23：使用新建的代码助手快捷方式

还可以以其他方式自定义代码助手。例如，在新建一个 Java 代码文件时，将自动插入下列注释：

```
/*
 * Created on Feb 4, 2004
 */
```



```
* To change the template for this generated file go to  
* Window>Preferences>Java>Code Generation>Code and Comments  
*/
```

通过选择 Window → Preferences → Java → Code Generation → Code and Comments → Code → New Java Files，可以改变上述注释中的代码，如图 3-24 所示。然后，可以编辑用于创建上述注释的模板，使其以需要的方式显示。

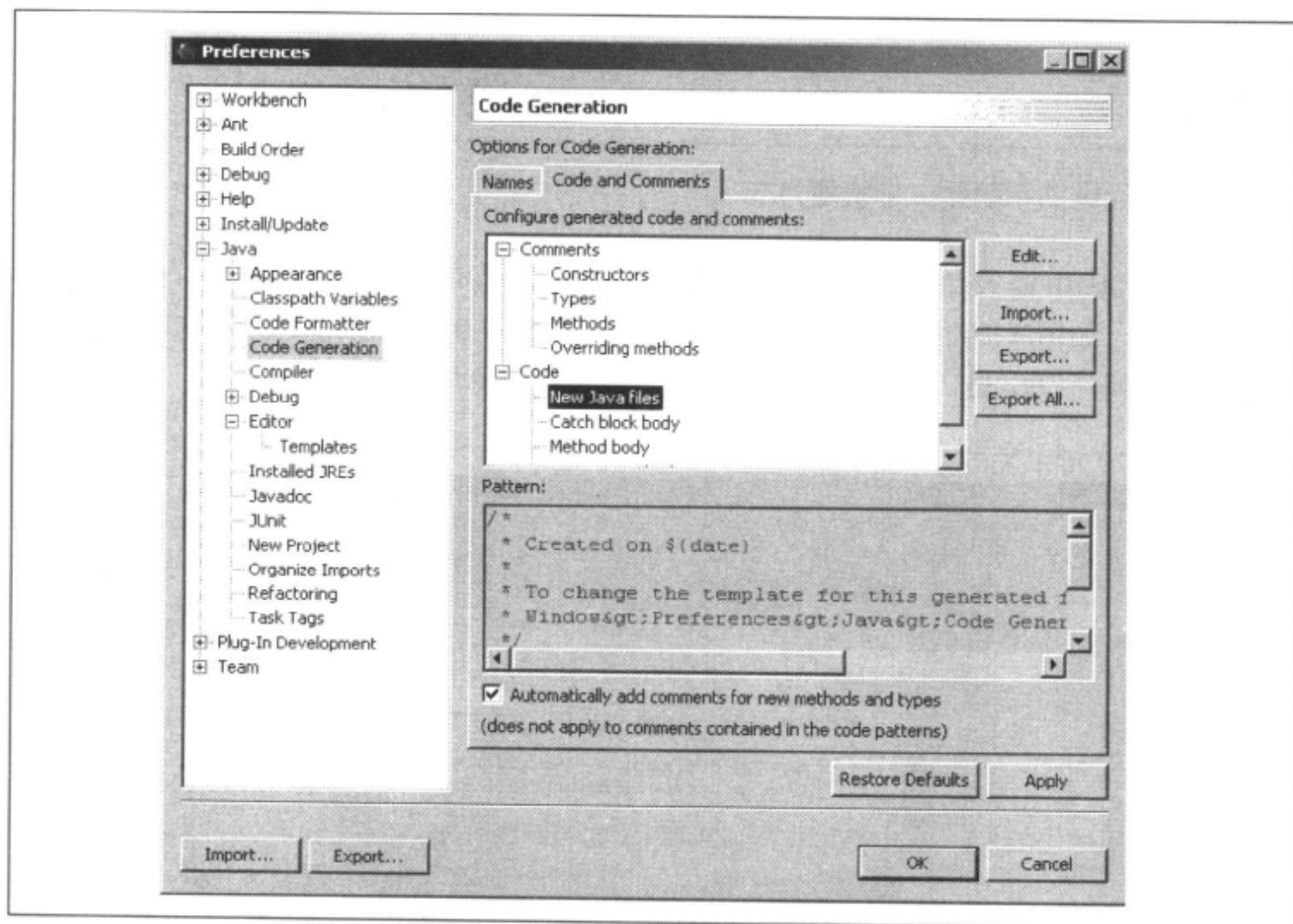


图 3-24：配置代码助手注释

注意： 还可以通过制定代码助手要创建的字段、静态字段、参数和本地辩论的前缀或后缀，来自定义代码助手。选择 Window → Preferences → Java → Code Generation → Names，然后设置代码助手、快速修复和重构（refactoring）中使用的变量的前缀和后缀。

Eclipse 3.0

与大家所期待的一样，在 Eclipse 3.0 可以以各种方式自定义代码助手。例如，可以自定义 getter/setter 方法的代码，还可以自定义插入新的字段时所采用的注释。

重构、编译和运行

4.0 简介

本章继续介绍关于JDT的内容，探索重构等操作以及应用程序的搜索、编译（build）和运行。你每天都可能使用Eclipse执行这些操作，这些都属于基本技能。

从本章开始，我们将介绍在Eclipse中如何处理重构，特别是如何重命名和移动元素。使用Eclipse这样好的Java IDE的主要好处之一是，当重命名和移动Java元素时，IDE可以自动更新整个代码对这些元素的所有引用。

除了重命名和移动元素以外，Eclipse还支持许多其他的重构操作。下面是这些操作的完整列表：

- 重命名元素
- 移动元素
- 修改方法的签名
- 将匿名类转换为嵌套类
- 将嵌套类型转换为顶级类型
- 降低或提升嵌套层次中的元素
- 将方法和静态字段转换为内联的
- 将局部变量转换为字段
- 提取方法、变量或常量
- 封装字段

在编写代码时，如果知道这些操作的原理及其作用，将是非常有用的。

Eclipse 3.0

除了上述列表以外，Eclipse 3.0 增加了下面的重构选项：

- 将成员移动到一个新的文件中
- 将类型通用化
- 创建工厂方法

4.1 重命名元素

问题

你想重命名代码中的变量、方法或其他元素，而你想确定该元素出现的每一个地方都没有被遗漏。

解决方案

在 JDT 编辑器中选择该元素，然后选择 Refactor → Rename，或者右击该元素，并选择 Refactor → Rename。然后输入该元素的新名称就可以了。

讨论

假如有如例 4-1 所示的代码，你认为 main 方法中的变量名 msg 过于简练，决定将其重新命名为 message。

例 4-1：一个简单的 main 方法

```
package org.cookbook.ch04;

public class Messenger
{
    public static void main(String[] args)
    {
        String msg = "No problem.";
        System.out.println(msg);
    }

    public static void printem(String msg)
    {
        System.out.println(msg);
    }
}
```

要重命名 `msg` 变量的所有引用，可选中该变量，并选择 **Refactor** → **Rename**，或者右击该变量，并选择 **Refactor** → **Rename**，打开如图 4-1 所示的对话框。

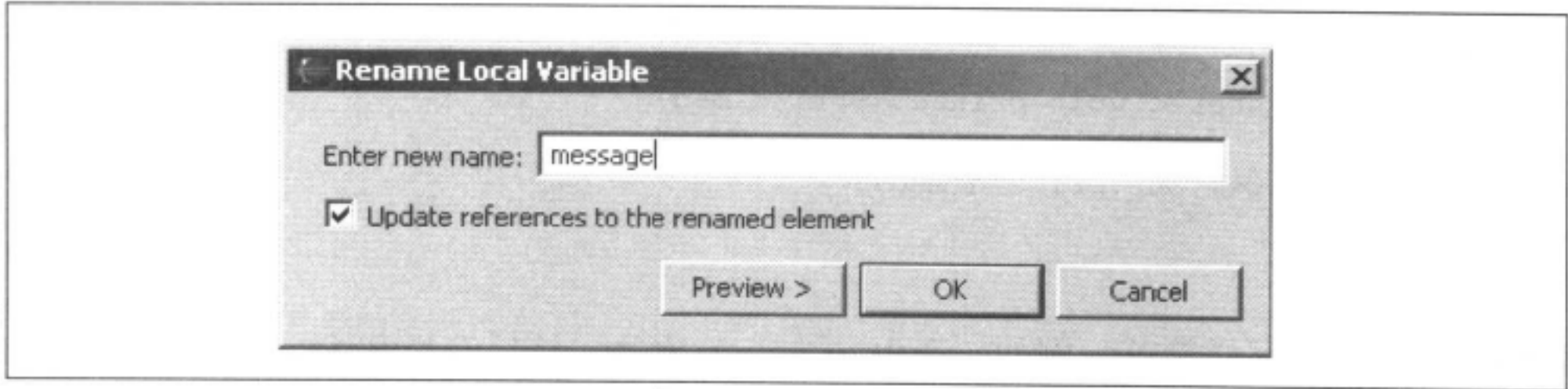


图 4-1：重命名一个局部变量

要将变量 `msg` 重命名为 `message`，在该对话框中输入单词“message”，然后单击 **Preview**，对所做修改进行预览，如图 4-2 所示。

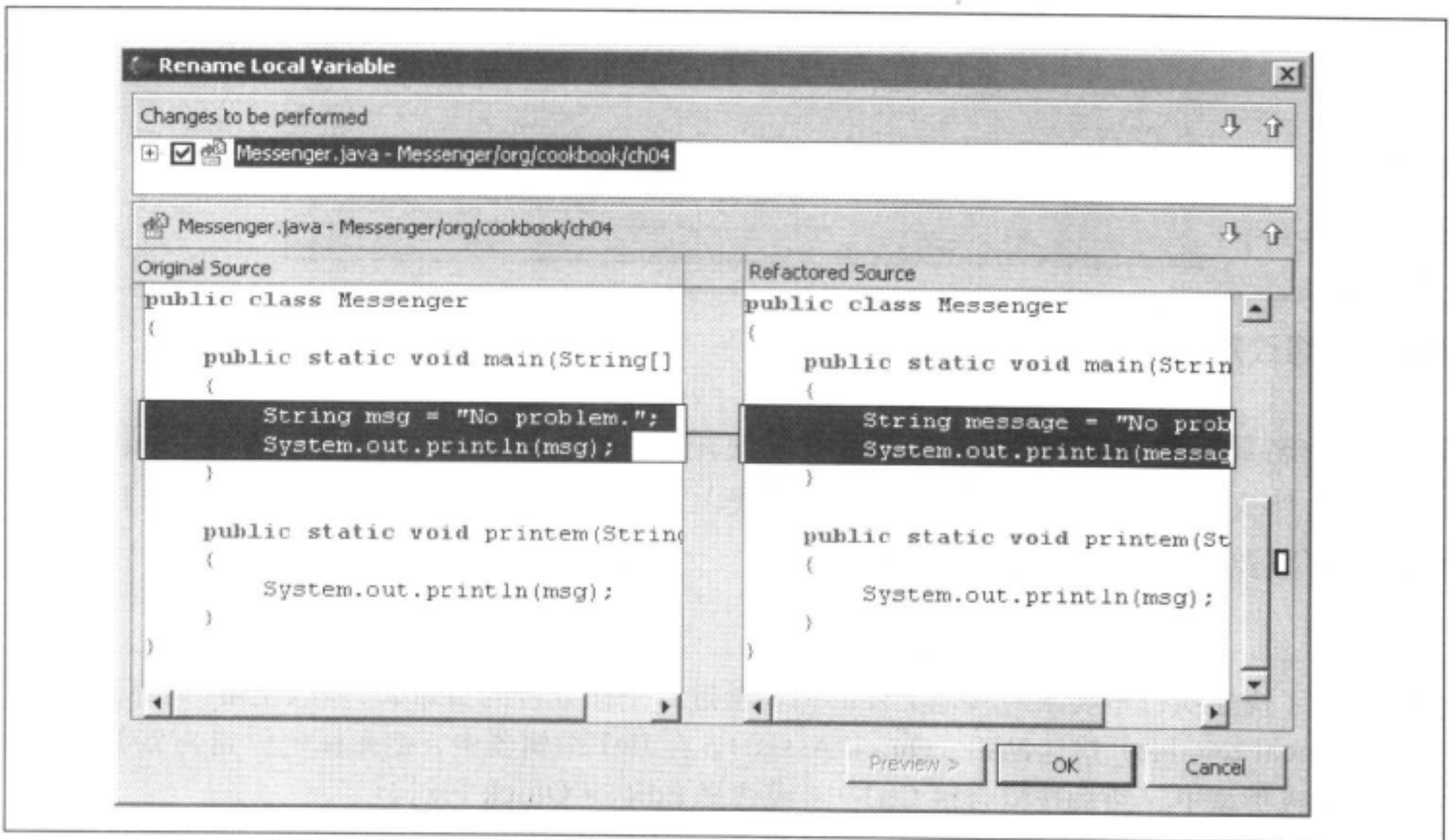


图 4-2：预览重构后的变化

Eclipse 是智能化的，所以它只修改对重命名变量的引用，而不会修改 `printem` 方法中无关的同名变量。单击 **OK**，以修改例 4-1 中的代码，使其如下所示：

```
package org.cookbook.ch04;

public class Messenger
{
```



```
public static void main(String[] args)
{
    String message = "No problem.";
    System.out.println(message);
}

public static void printem(String msg)
{
    System.out.println(msg);
}
```

重构一个方法名同样简单。例如，选中 `printem` 方法，然后选择 **Refactor** → **Rename**，或者右击该方法，然后选择 **Refactor** → **Rename**。这将打开 **Rename Method** 对话框，如图 4-3 所示，在此我们将该方法重命名为 `display`。单击 **OK**，即可重命名该方法及其所有引用。

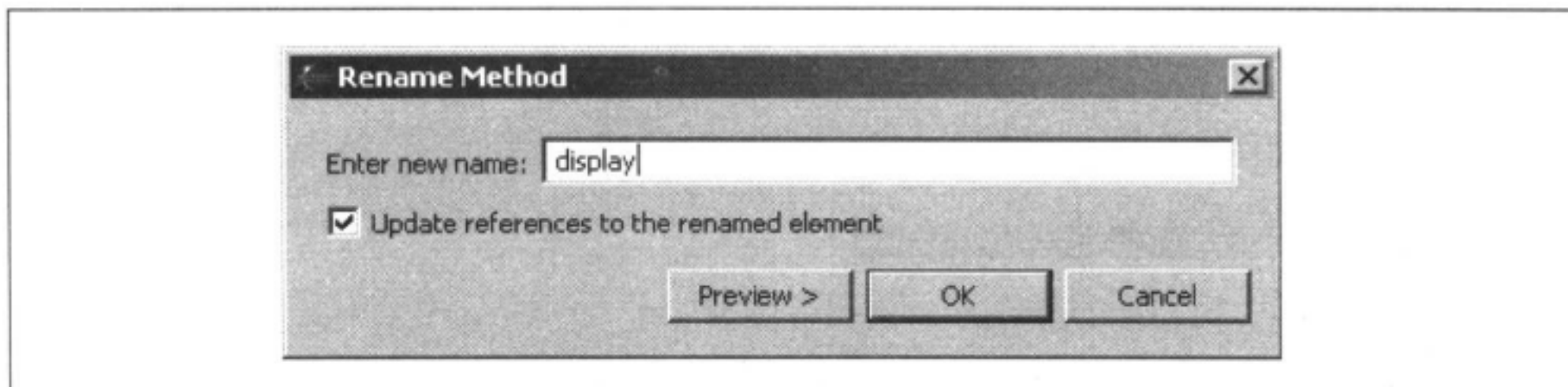


图 4-3：修改方法名

除了重命名局部变量和方法以外，还可以使用 Eclipse 的 **Refactor** 菜单重命名项目、资源、原文件夹、包、编译单元、类型（如类）、字段、方法和参数。还应注意的是，重构操作可以跨越多个文件自动进行。

注意： 为了快速执行不要求对其他文件的相关性进行全面分析的重命名，可以使用“局部重命名”（local rename）快速助手（Quick Assist）。在 JDT 编辑器中，把光标定位到一个变量、方法或类型中，然后按组合键 **Ctrl-I**（或选择 **Edit** → **Quick Fix**）。

Eclipse 3.0

在 Eclipse 3.0 中，只要选中相应的复选框，不仅可以更新代码中对重命名元素的引用，而且可以更新串文字、注释、Javadoc 注释甚至非 Java 文件中的引用，如图 4-4 所示。

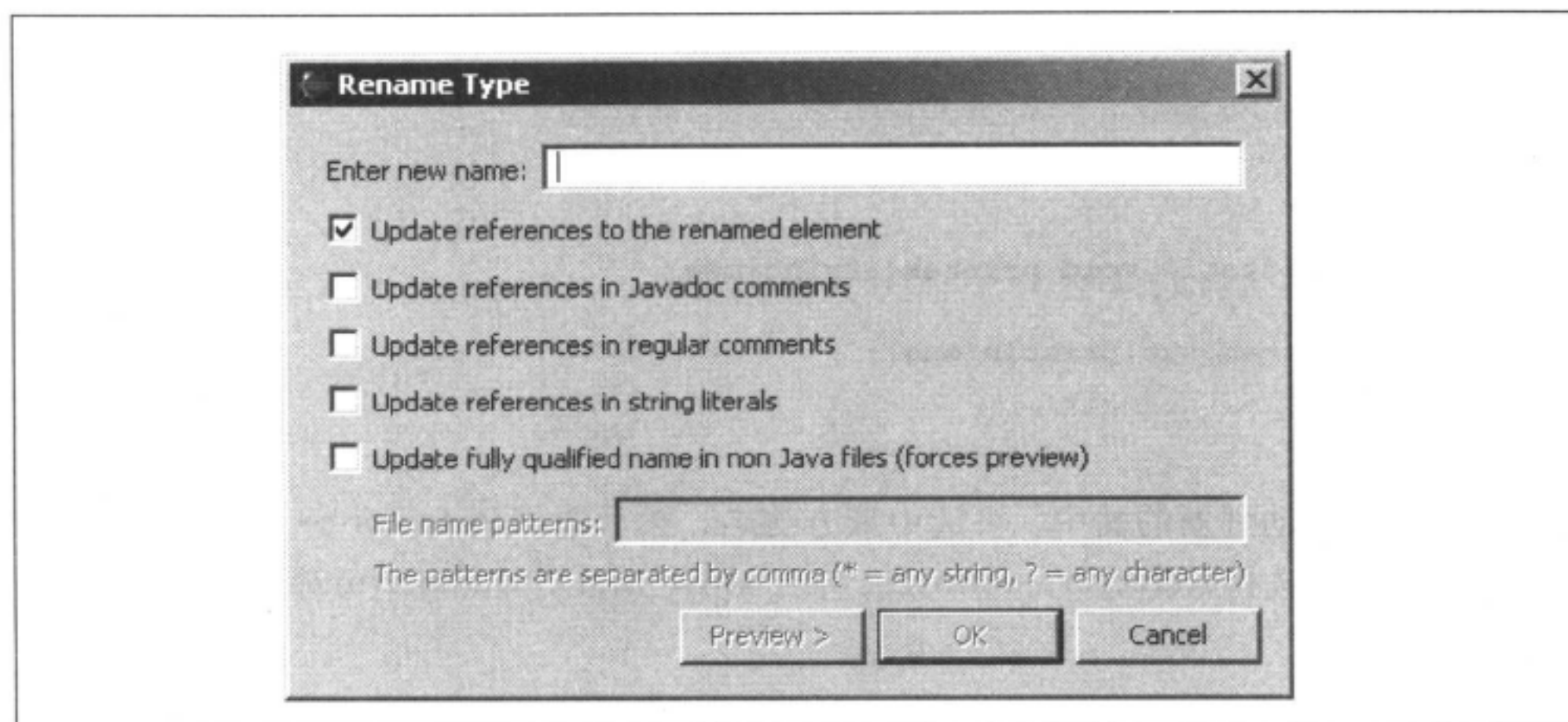


图 4-4: Eclipse 3.0 中的重命名选项

参考

4.2 节，移动元素；4.7 节，从本地记录恢复元素和文件；*Eclipse* (O'Reilly) 一书的第 2 章。

4.2 移动元素

问题

你想通过在类、方法、包或其他容器之间移动元素，来重新组织代码。

解决方案

选择要移动的元素，然后选择 Refactor → Move，或者右击要移动的元素，并选择 Refactor → Move。然后使用 Move 对话框移动元素。

讨论

假如你想把 `printem` 方法，从当前类 `Messenger`，移动到项目中的另一个类 `Messenger2` 中：

```
package org.cookbook.ch04;

public class Messenger
{
```

```
public static void main(String[] args)
{
    String msg = "No problem.";
    System.out.println(msg);
}

public static void printem(String msg)
{
    System.out.println(msg);
}
```

要把这个方法移动到新的类中，可选中该方法名，然后选择 Refactor → Move，或者右击该方法名，并选择 Refactor → Move，打开 Move Static Member(s)对话框，如图 4-5 所示。

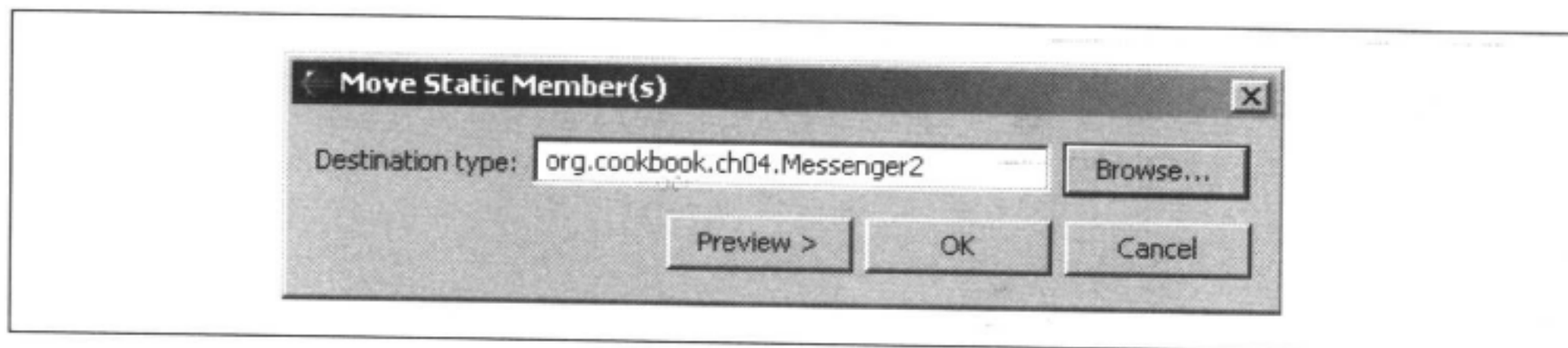


图 4-5：选择一个目标类型

输入方法要移动至的类的完全限定名称“org.cookbook.ch04.Messenger2”，或浏览到该类，然后单击 OK。该方法将被移动到新的类中：

```
package org.cookbook.ch04;

public class Messenger2
{
    public static void main(String[] args)
    {
    }

    public static void printem(String msg)
    {
        System.out.println(msg);
    }
}
```

对被移动方法的所有引用也将自动更新，所以代码没有被搅乱：

```
package org.cookbook.ch04;

public class Messenger
{
```

```
public static void main(String[] args)
{
    String message = "No problem.";
    Messenger2.printem(message);
}
}
```

在 Eclipse 中，可以通过重构移动静态方法、静态字段或实例方法。

注意： 可以通过拖放的方式，在包之间移动任何 Java 编译单元。所有遗漏的导入将自动添加，而引用将自动更新。

参考

4.6 节，根据本地历史记录比较文件；4.7 节，从本地历史记录恢复元素和文件；*Eclipse* (O'Reilly) 一书的第 2 章。

4.3 接口的提取与实现

问题

你想从类中提取接口。

解决方案

选中一个类名，然后选择 Refactor → Extract Interface，或者右击该类名，然后选择 Refactor → Extract Interface。输入要提取的接口名称，选择要在该接口中声明的成员，并单击 OK。

讨论

Eclipse 的重构特性使你能够从类中提取接口。例如，假如有例 4-1 所示的代码，其中 Interfaces 类包括非静态的 printem 方法。

例 4-2：要从中提取接口的一个类

```
package org.cookbook.ch04;

public class Interfaces
{
```



```
public static void main(String[] args)
{
    String msg = "No problem.";
    new Interfaces().printem(msg);
}

public void printem(String msg)
{
    System.out.println(msg);
}
}
```

提取接口

要从 `Interfaces` 类中提取接口，在代码中右击该类的名称，选择 `Refactor` → `Extract Interface`，这将打开如图 4-6 所示的对话框。选中 `printem` 方法，输入名称“`NewInterface`”，并单击 `OK`。

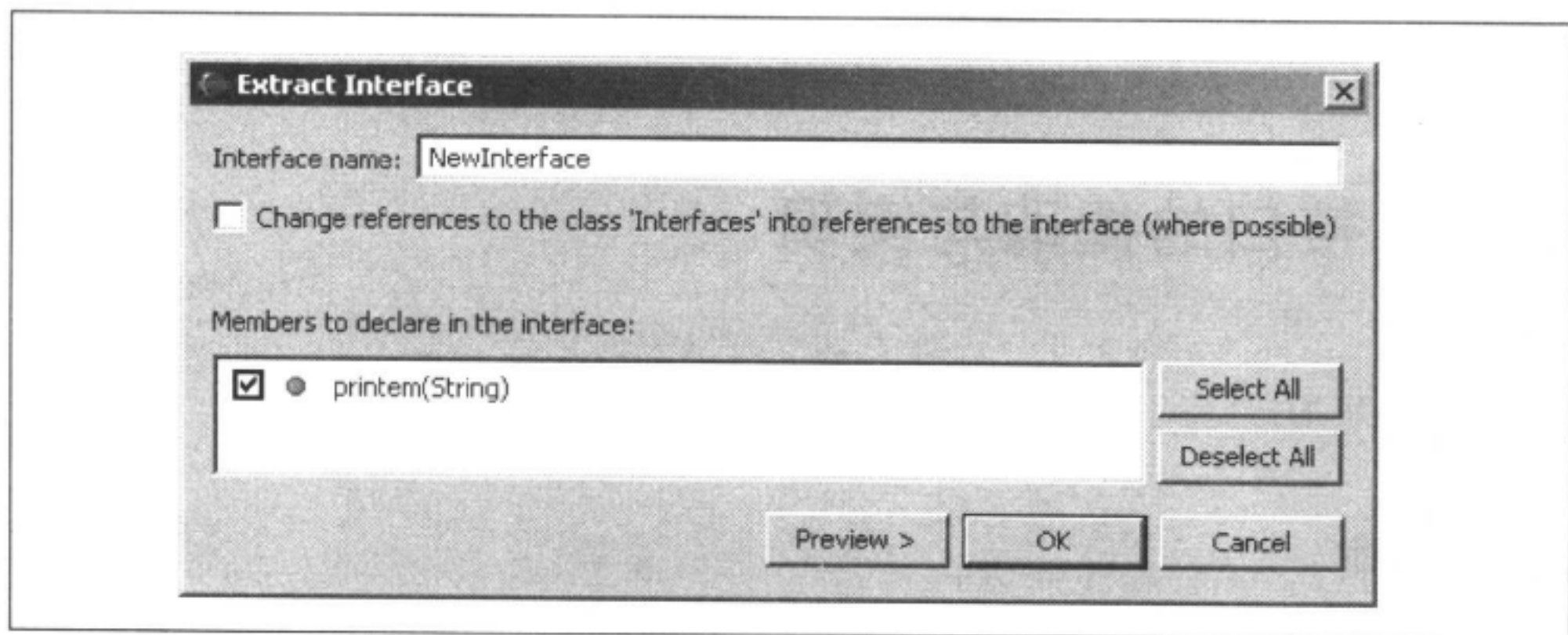


图 4-6：提取接口

这将用新建的接口创建一个新的文件，即 `NewInterface.java`：

```
package org.cookbook.ch04;

public interface NewInterface
{
    public abstract void printem(String msg);
}
```

注意：也可以选择 `File` → `New` → `Interface`，从头创建新的接口。

实现接口

通过输入 `implements` 关键字和要实现的接口的名称（如 `public class ServletExample implements NewInterface`），可以实现接口。JDT 编辑器显示一个 Quick Fix 灯泡图标，表示方法缺失。单击该灯泡图标（也可以按组合键 `Ctrl-1` 或选择 `Edit → Quick Fix`），让 Eclipse 实现缺失的方法。

注意：不必实现接口的所有方法。Eclipse 允许选择生成类抽象。

4.4 搜索代码

问题

你想搜索项目中的某个文件或全部文件，看其中是否包含匹配的文本。

解决方案

使用 Eclipse 内置的 Search 对话框。Search 菜单包含多个菜单项（`Search → Search`，`Search → File`，`Search → Help`，`Search → Java`），但所有这些菜单项均打开同一对话框。

讨论

正如你对一个好的 IDE 所期待的一样，Eclipse 内置了大量的搜索支持。然而，Eclipse 中的搜索不是通过标准的 `Edit → Find/Replace` 操作实现的，这种操作只能在当前文件中进行。Eclipse 的搜索在 Search 视图中显示所有匹配结果，可以选择跳转到其中一个或多个搜索结果。另外，Search 菜单中的菜单项可以搜索工作区中的所有文件、整个帮助系统、所有插件，等等。还可以使用通配符（* 和 ?）。

Search 对话框如图 4-7 所示；注意其中的 4 个标签页：

File Search

在工作集、工作区或所选的文件中，搜索指定文件。

Help Search

搜索帮助系统中的匹配文本。

Java Search

搜索工作区、工作集或所选的文件中的匹配文本。可以指定搜索的类型、方法、包、构造函数或字段。

Plug-in Search

搜索插件、程序段（fragment）和扩展点（extension point）。

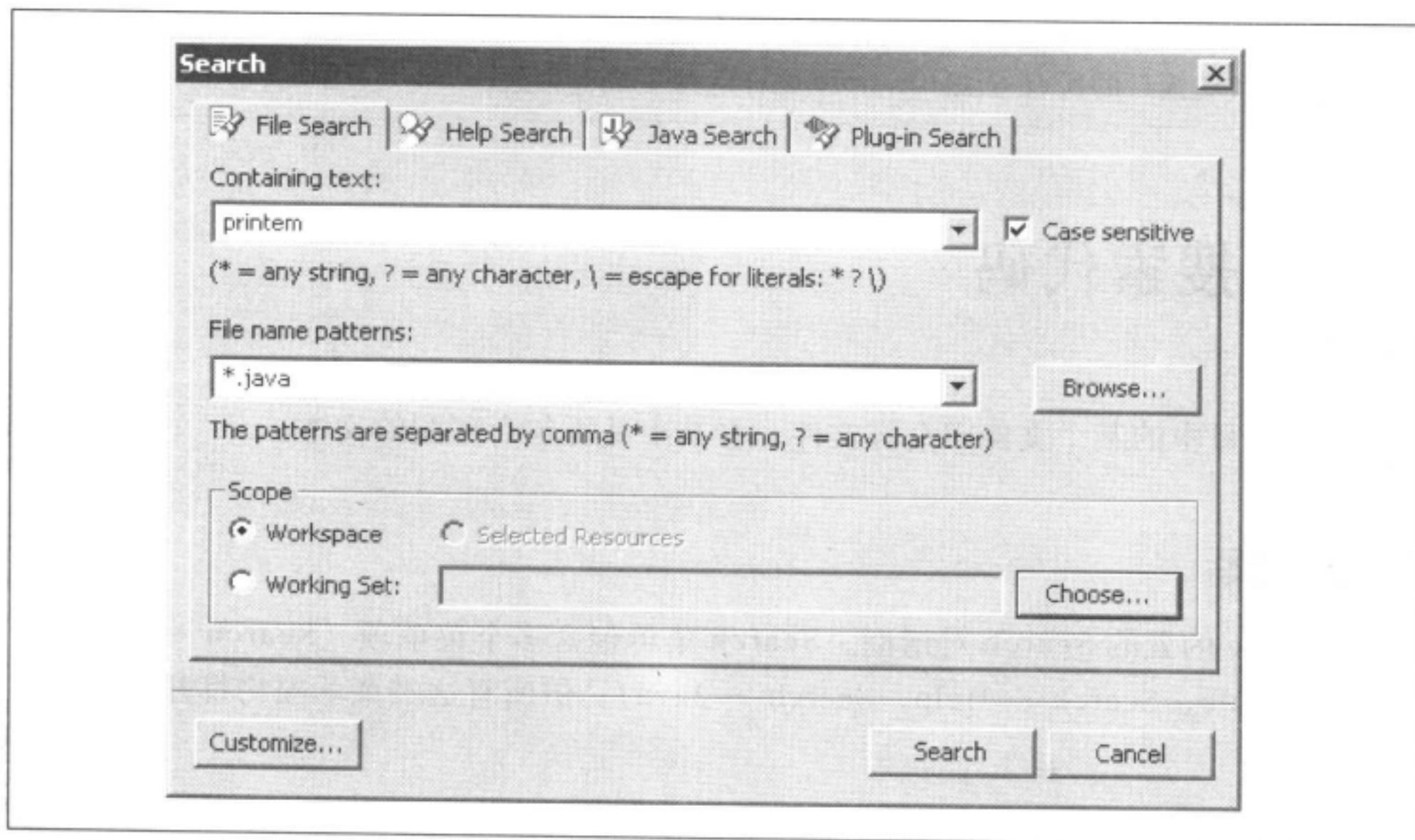


图 4-7: Search 对话框

例如，看一下 File Search 标签页，通过它可以搜索多个文件中的指定文本（在本例中是 printem）。在 File name patterns 文本框中，可以指定要搜索的文件的类型，如 "*.java, *.*" 等等，而且利用该文本框下面的单选按钮可以指定搜索的范围。

注意：在此对话框中，默认情况下，Scope 选项组中的 Selected Resources 单项按钮没有启用。如果只想搜索当前项目或数目有限的没有使用工作集的项目，可在 Package Explorer 视图中选中当前项目或限定的项目，然后打开 Search 对话框，并单击 Selected Resources 单项按钮。

单击 Search 按钮后，搜索结果将出现在 Search 视图中，如图 4-8 所示。双击一个匹配条目，可在 JDT 编辑器中将其打开，且在标记栏中出现一个箭头，如图 4-8 所示。

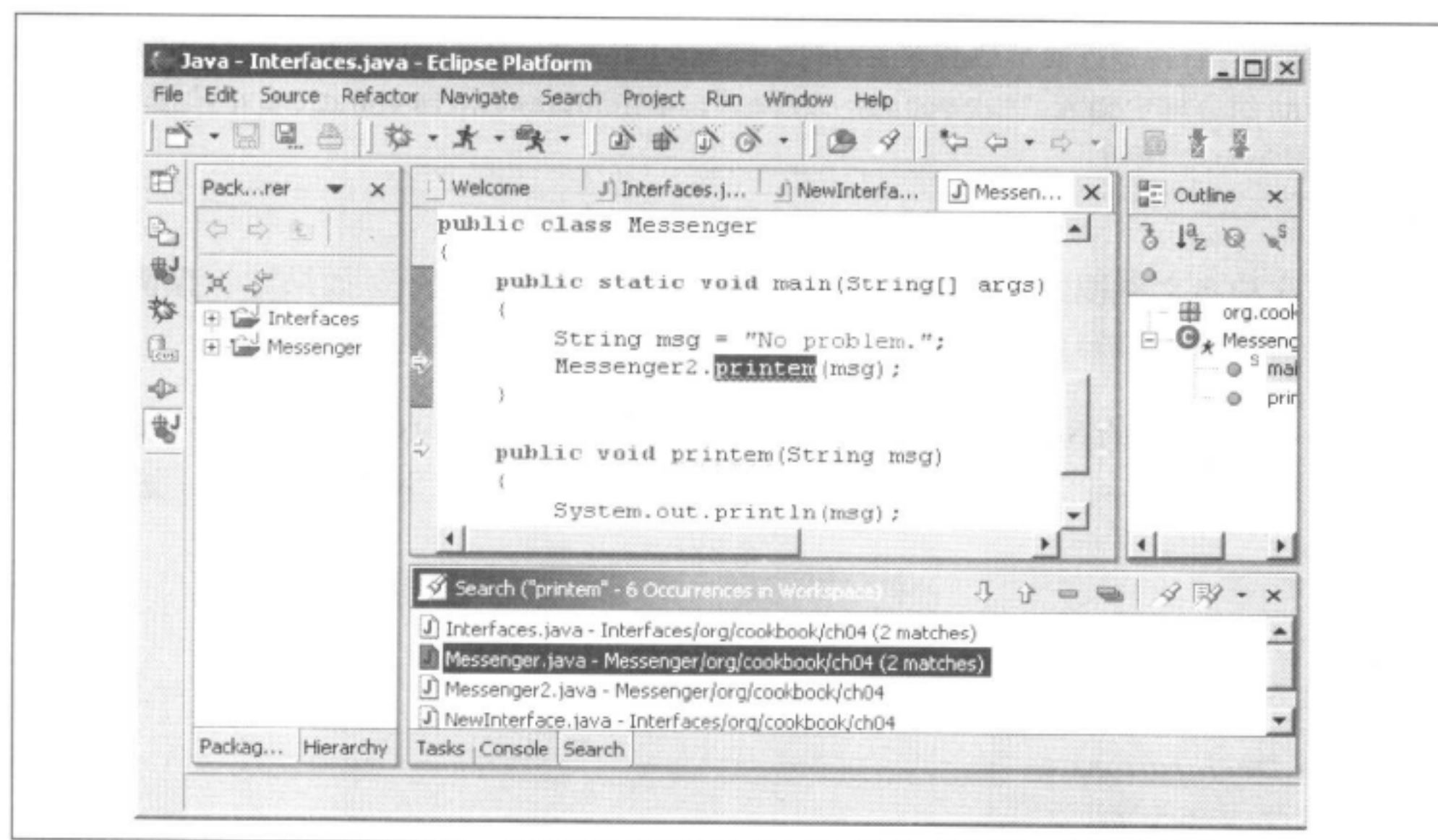


图 4-8：搜索结果

使用 Search 对话框中的 Java Search 标签页，还可以进行 Java 搜索。这种搜索可以按种类——类型、方法、包、构造函数和字段——搜索 Java 元素，如图 4-9 所示。还可以对搜索进行限制，使其仅搜索引用中的匹配条目，如图 4-9 所示。

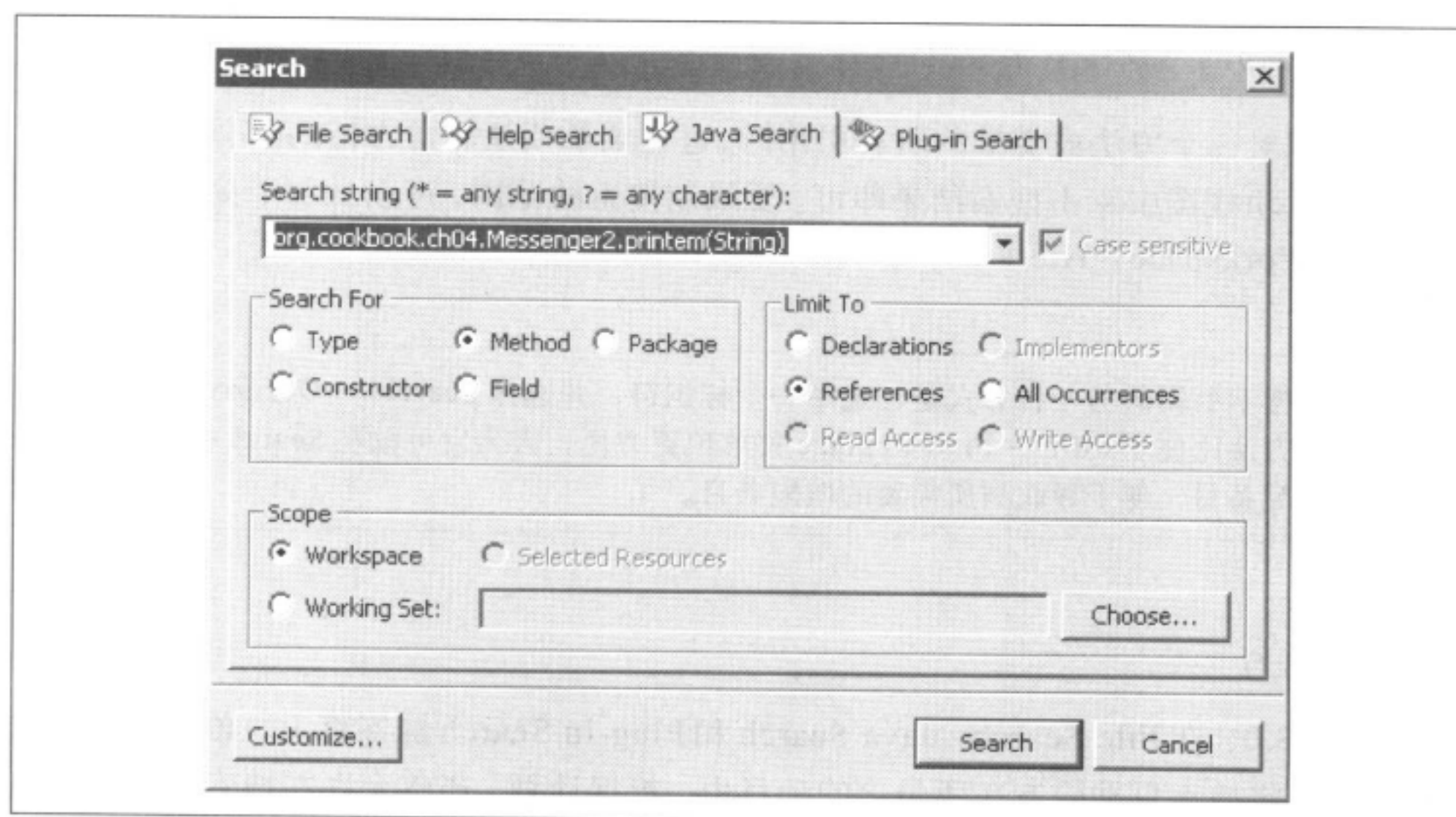


图 4-9：执行 Java 搜索

注意：要搜索具有特定返回类型的方法，可打开Search对话框，并单击Java Search标签。在Search string 组合框中输入“* <return type>”，单击Method 和Declarations 单项按钮，然后单击Search 按钮。

能够进行这种多文件搜索，是人们使用一种IDE的主要原因之一。如果你一直使用一种简单的文本编辑器编写Java代码，你会发现两者不可同日而语。

除了Search 菜单中的Search → Search、Search → File、Search → Help 和Search → Java 菜单项以外，还可以选中代码中的一个元素，并选用下列Search 菜单项，进行快速搜索：

Search → References

搜索对元素的引用。

Search → Declarations

搜索元素的声明。

Search → Implementors

搜索元素的实现程序。

Search → Read Access

搜索读访问（仅搜索字段）。

Search → Write Access

搜索写访问（仅搜索字段）。

当需要跟踪对一个方法或变量的所有引用时，这些菜单项是非常方便的。只需搜索引用，然后在Search 视图中双击搜索结果即可。要限制搜索的范围，可以从每个菜单项的子菜单中选择Workspace、Hierarchy 或Working Set。

注意：快捷搜索引用的另一种方式是，选中一个标识符，并选择Search → Occurrences in File。这种方法比使用Edit → Find/Replace 菜单项更方便，因为它可以在Search 视图中列出所有匹配条目，便于导航到所需要的匹配条目。

Eclipse 3.0

在Eclipse 3.0，在File Search、Java Search 和Plug-in Search 标签页上，单击一个单选按钮，即可将搜索自动限制在所包含的项目中。根据计划，还将允许在搜索中使用正则表达式。

4.5 比较文件

问题

你想以图形显示的方式查看两个文件的差别。

解决方案

在一个视图中选中要比较的文件，然后在视图的快捷菜单中选择 Compare With → Each Other。

讨论

专业开发人员经常要做的事情之一是比较两个文件的内容，以查看对每个文件的修改；大多数操作系统对这种操作只提供最低限度的支持。

要比较两个文件，可以在一个视图（如 Package Explorer 视图）中选中它们，然后在该视图的快捷菜单中选择 Compare With → Each Other。Eclipse 将用智能的方式进行比较，并在 Compare Editor 视图中显示比较结果，如图 4-10 所示。

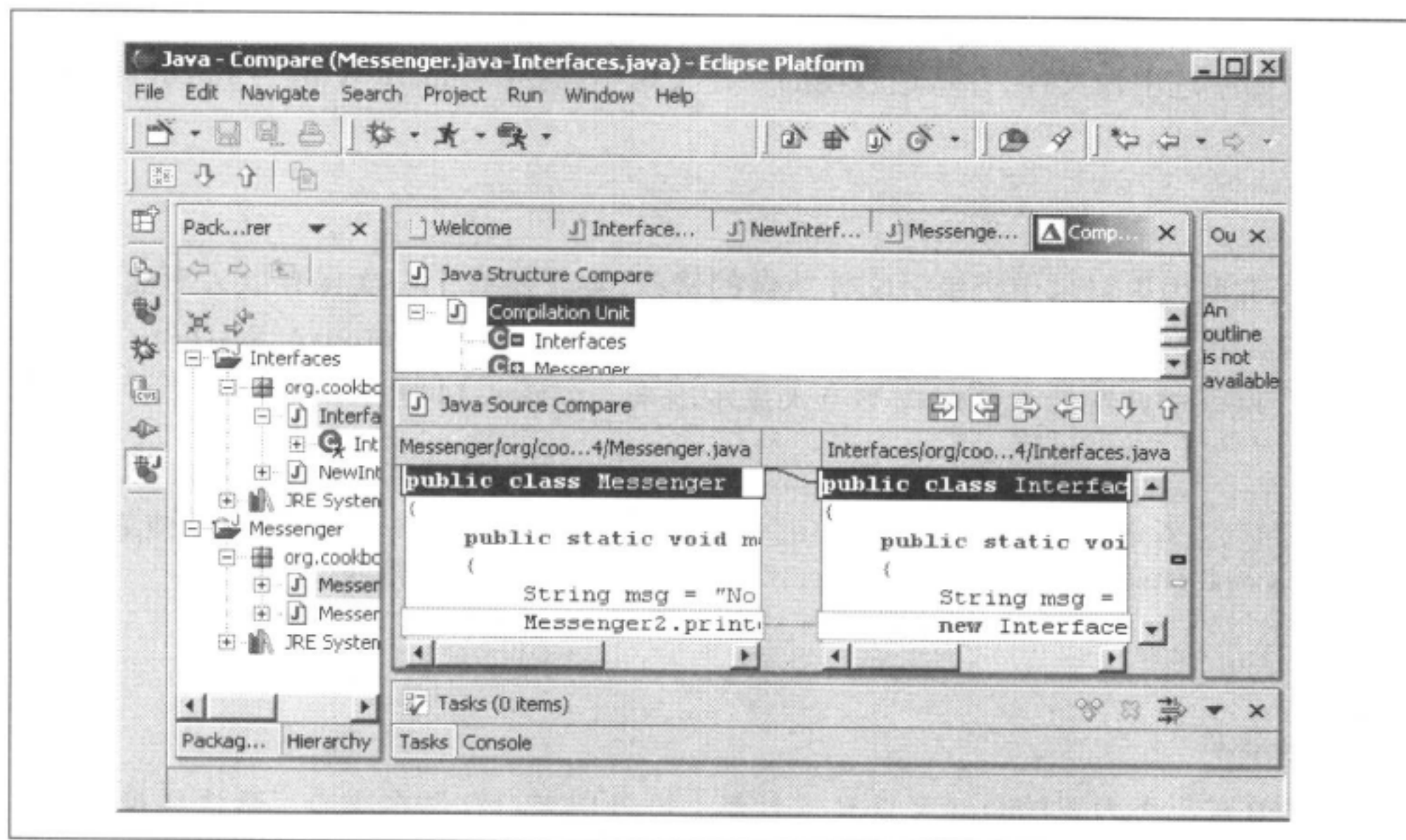


图 4-10：比较两个文件

甚至可以编辑文件，使用 Java Source Compare 中的按钮将修改从一个文件复制到另一个文件中。

注意： 在进行比较时可以忽略注释和格式的变化：在 Compare Editor 工具栏按钮（主工具栏上右数第4个按钮，如图4-10所示）中单击 Ignore Whitespace 选项。当使用 XML 时，这种功能是非常有用的。

参考

4.6 节，根据本地历史记录比较文件；4.7 节，根据本地历史记录恢复元素和文件。

4.6 根据本地历史记录比较文件

问题

你想查看自上一次保存以来对文件做了什么修改。

解决方案

在一个视图中选中该文件，并从该视图的快捷菜单中选择 Compare With → Local History。

讨论

Eclipse 在本地历史记录中记录对文件所做的修改。要了解自上一次保存以来对文件做了什么修改，可以选中该文件并右击它，从快捷菜单中选择 Compare With → Local History。最近所做的修改将以图形方式显示出来，如图4-11所示。

注意： 还可以查看单个元素的最近的历史记录（不是整个文件）。右击一个元素，从快捷菜单中选择 Local History → Compare With，即可以图形方式查看最近的修改。

Eclipse 3.0

Eclipse 3.0 有一个内置的 Quick Diff 工具栏，可以用来对存储在磁盘（默认磁盘）上的文件进行逐行比较，还可以对 CVS 储存库中的修改进行比较。在图4-12中可以看到，紧靠 JDT 编辑器左侧的就是 Quick Diff 工具栏。要将一行与磁盘上该文件的最新版本进行比较，只需将鼠标指针停留在 Quick Diff 工具栏上即可。

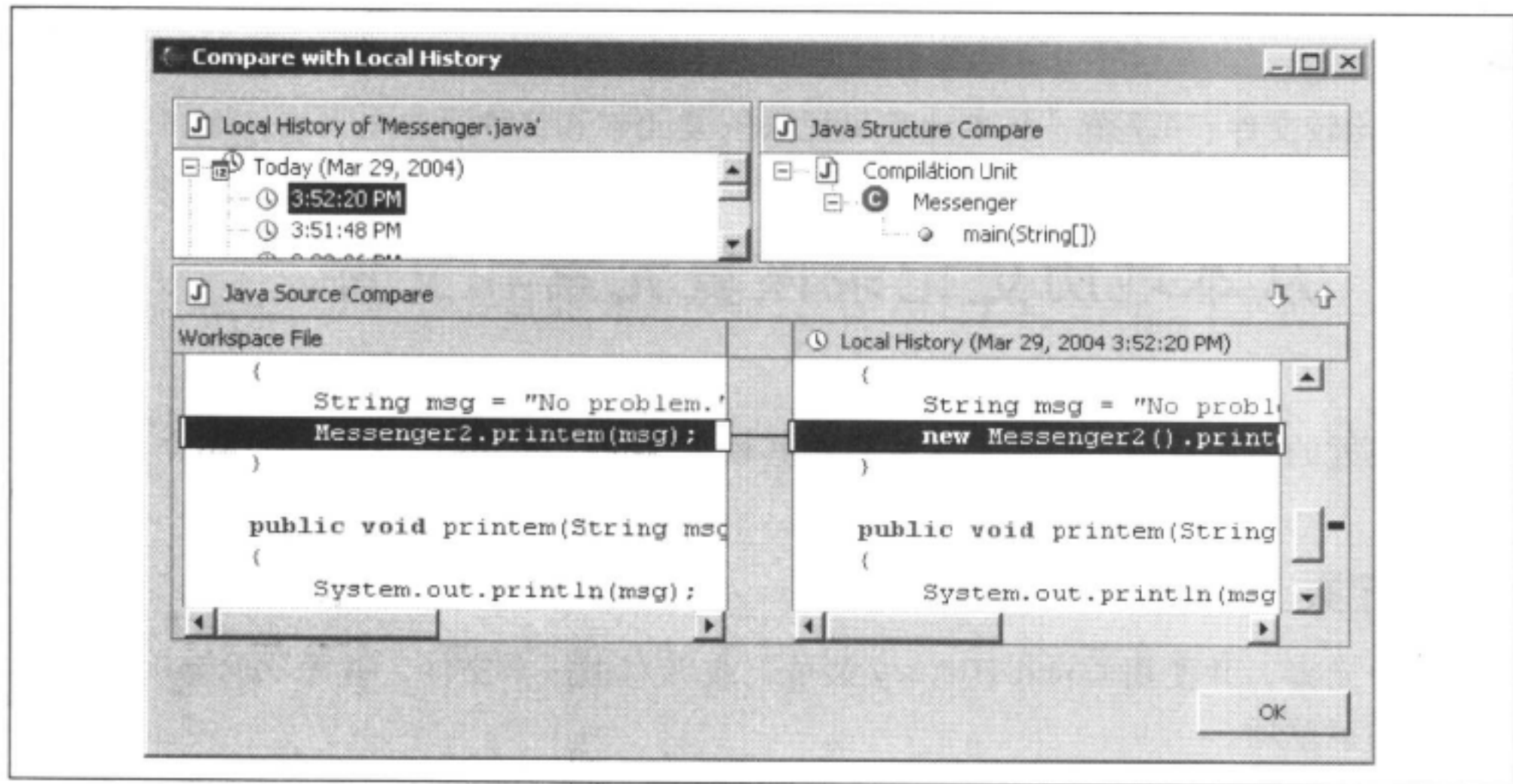


图 4-11: 根据本地历史记录比较文件

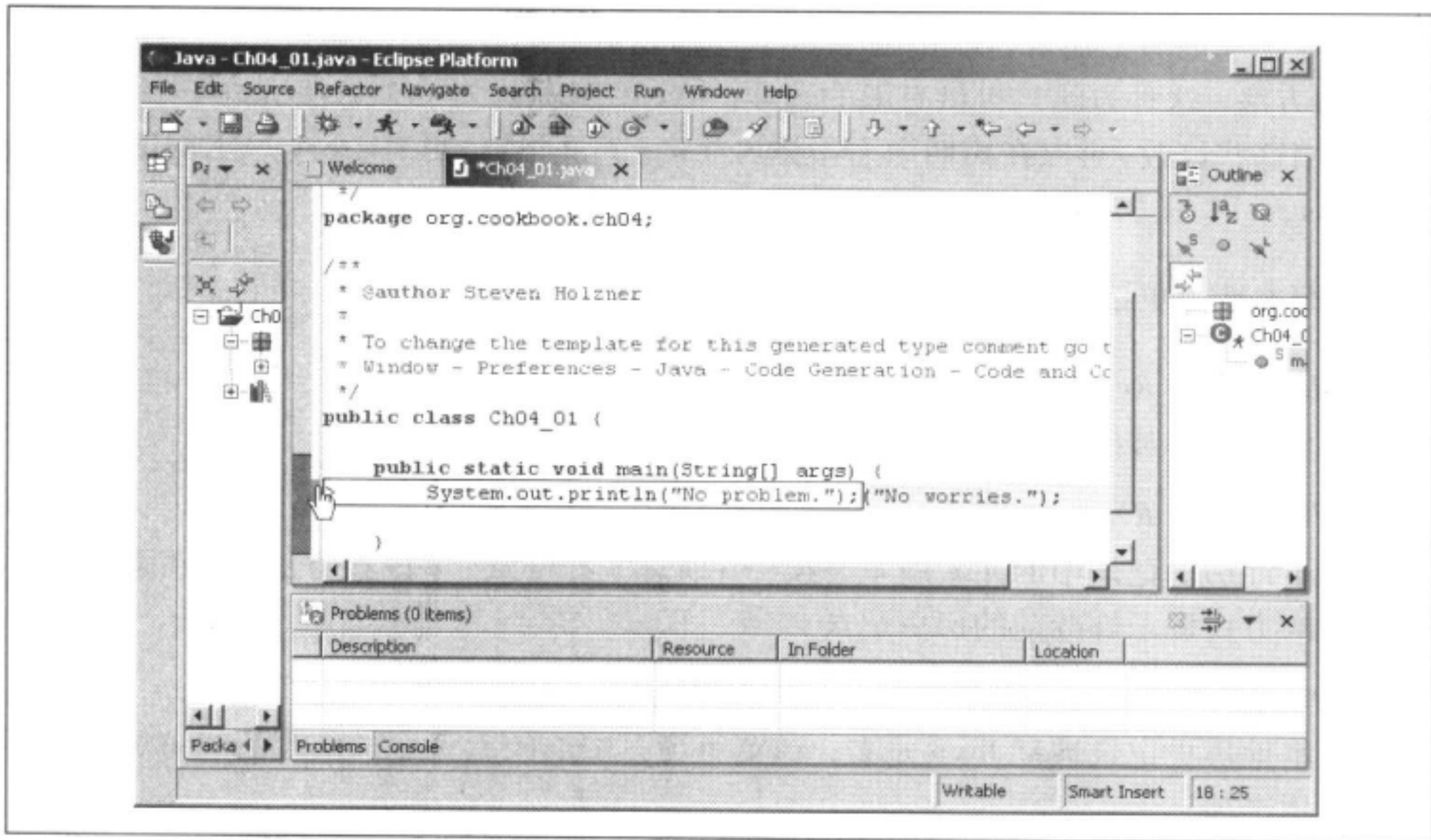


图 4-12: Eclipse 3.0 的 Quick Diff 工具栏

注意：要关闭Quick Diff，右击Quick Diff工具栏，然后选择Disable QuickDiff即可。要使Quick Diff根据CVS储存库中的代码进行比较，可右击Quick Diff工具栏，然后选择Set QuickDiff Reference → Latest CVS Revision。

参考

4.5 节，比较文件；4.7 节，从本地历史记录恢复元素和文件。

4.7 从本地历史记录恢复元素和文件问题

你确定最近的一次修改是错误的，并想把其恢复到前一版本。

解决方案

右击一个元素，并使用 Local History 菜单，或者右击一个文件，并使用 Replace With 菜单中的菜单项。

讨论

由于保存了本地历史记录，Eclipse 使你能够撤销最近的修改，而不必采用选择 Edit → Undo 的方法，这种方法只提供有限的选项。例如，如果你对一个方法进行了一些修改，并想撤销这些修改，可以在声明中右击该方法名，并选择 Local History → Replace With Previous。

可以根据本地历史记录恢复文件的 JDT 编辑器的快捷菜单项如下：

Local History → Replace With Previous

用上一版本替换文件。

Local History → Replace With

用本地历史记录中的元素版本替换一个元素，如变量、字段或方法。该菜单项以图形的方式显示修改前的历史记录。

Local History → Restore From

从本地历史记录恢复 Java 元素。该菜单项以图形的方式显示修改前的历史记录的可选项。

例如，选择 Local History → Replace With，可以回顾修改前的 Java 元素，如图 4-13 所示。

在一个视图（如 Package Explorer 视图）中右击文件，并选择下列菜单项之一，可以把文件恢复到其上一版本：

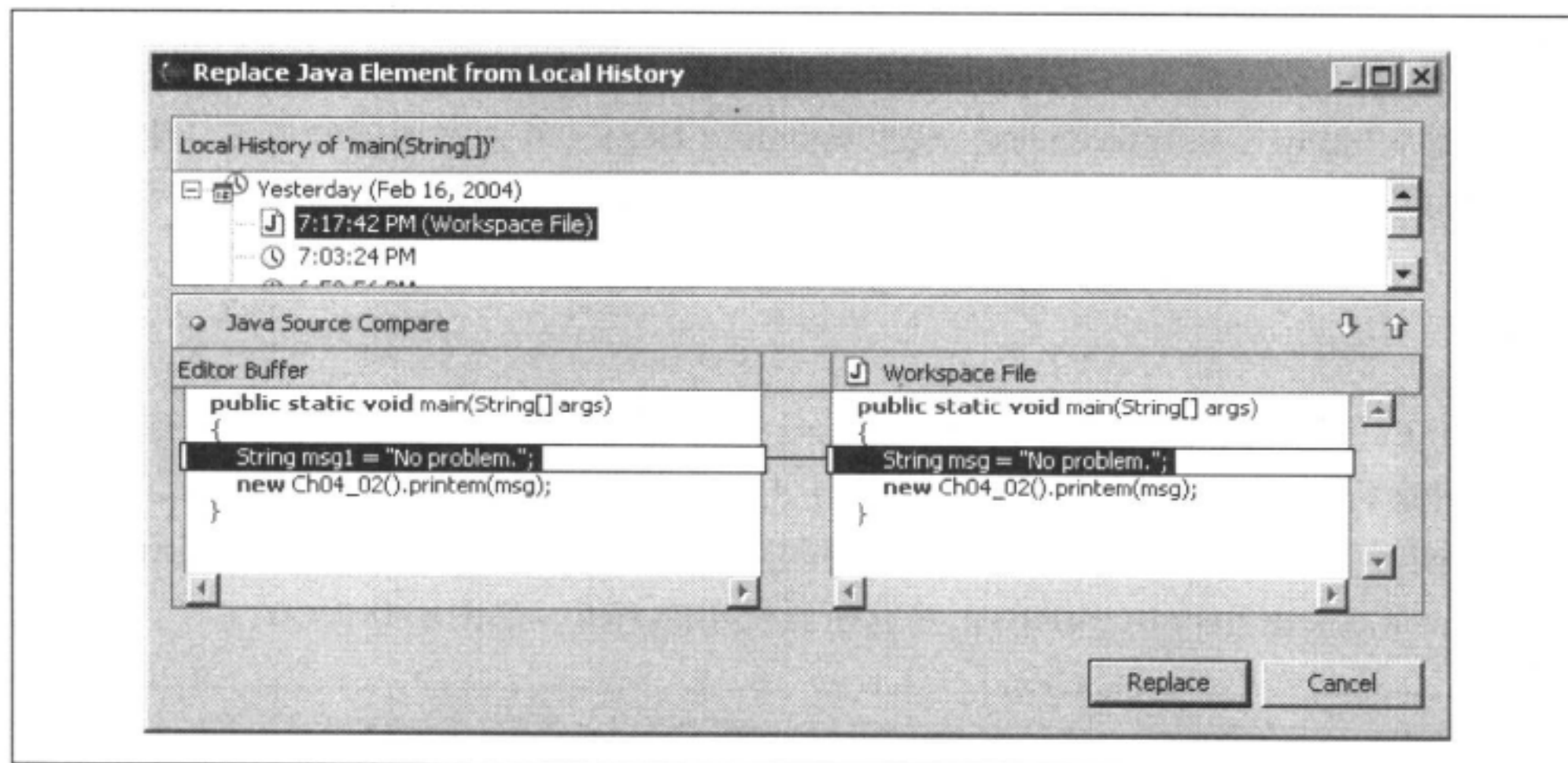


图 4-13：从本地历史记录恢复 Java 元素

Replace With → Previous from Local History

用本地历史记录中的文件的上一版本替换文件。

Replace With → Local History

用从本地历史记录中选择的版本替换文件。

Restore from Local History

从本地历史记录恢复文件。

Eclipse 3.0

在 Eclipse 3.0 中，可以将一行代码恢复到其初始状态。右击 Quick Diff 工具栏，并选择 Revert Line。

参考

4.5 节，比较文件；4.6 节，根据本地历史记录比较文件。

4.8 选择 Java 编译运行环境

问题

你想使代码以某种 JRE 作为目标运行环境，或者改变 Eclipse 使用的默认 JRE。

解决方案

通过选择 Window → Preferences, 单击 Installed JREs, 并选择所需的 JRE (使用 Add 按钮添加 JRE 和 SDK), 可以指定 Eclipse 使用的 Java 运行环境。

讨论

首次运行 Eclipse 时, 它将搜索已安装的 Java 运行环境, 这意味着它可能不使用你想使用的那种运行环境。例如, Eclipse 可能会使用与浏览器一起安装的过时的 JRE, 而不是使用你刚刚下载的新版 Java SDK。为了指定要使用的运行环境, 可选择 Window → Preferences, 单击 Installed JREs, 并选择要使用的 JRE, 如图 4-14 所示。

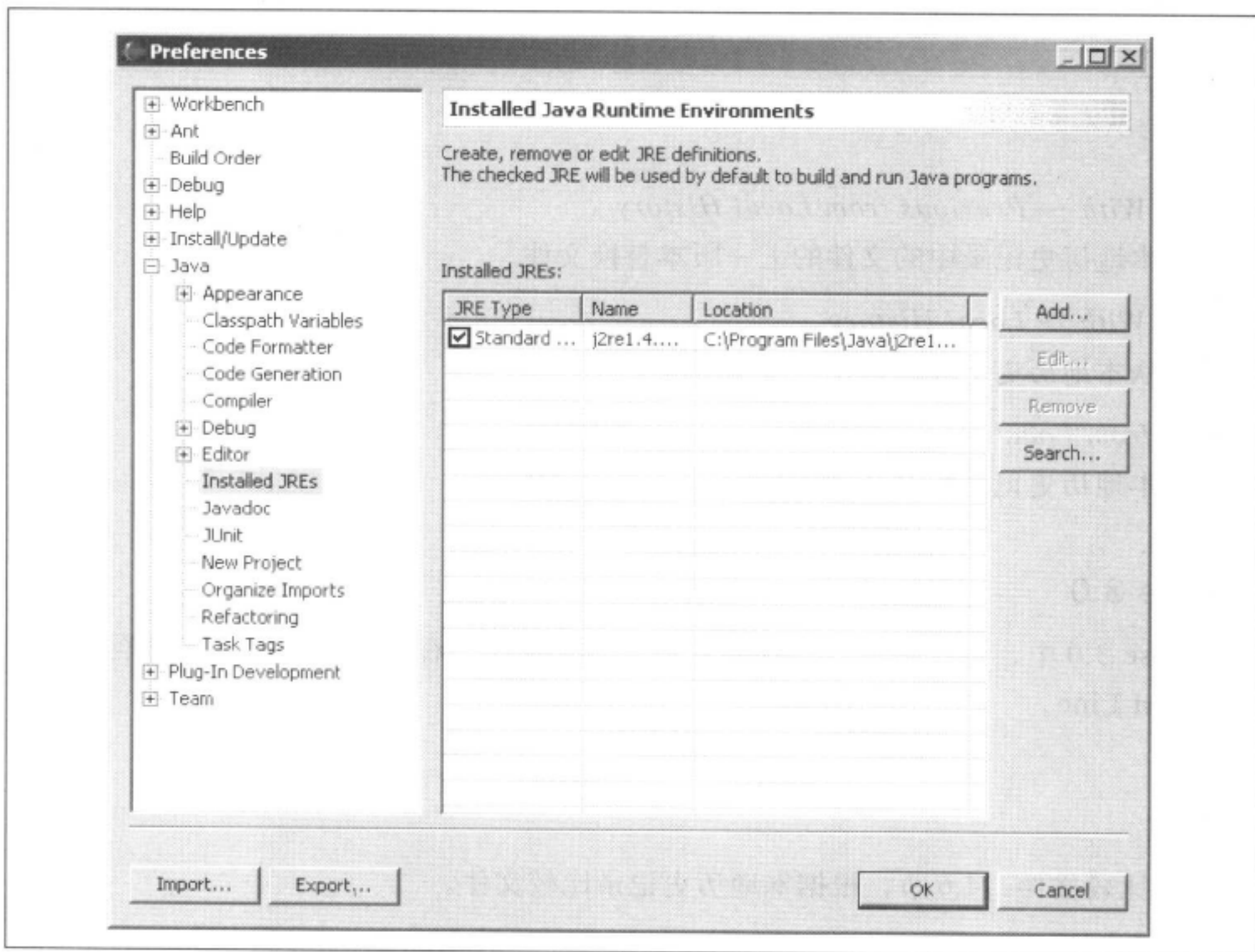


图 4-14: 选择一种 JRE

注意: 要设置项目特定的 JRE (当需要使一个项目以某种 JRE 为目标运行环境时), 可选择 Project → Properties → Java Build Path, 单击 Libraries 标签, 单击 JRE System Library, 并单击 Edit。在出现的对话框中, 选择项目特定的 JRE。

使用 -vm 开关设置 JVM

如果 JRE 的版本太老，Eclipse 可能不会启动。在这种情况下，可从命令行启动 Eclipse，并使用 -vm 开关，指定要使用的 JRE。例如，可以输入：

```
eclipse -vm c:\jre1.4\bin\javaw.exe
```

在 Eclipse 启动之后，可以设置 Eclipse 以后使用的 JRE。

参考

4.9 节，运行代码；*Eclipse* (O'Reilly) 一书的第 2 章。

4.9 运行代码

问题

你想运行代码。

解决方案

在 Run 菜单中选择以下菜单项之一：

Run → *Run Last Launched*

运行上一次运行的程序。

Run → *Run History*

通过此菜单项可以从子菜单中选择要运行的程序。

Run → *Run As*

通过此菜单项可以选择程序的运行方式——Java Applet、Java Application、JUnit Test（将在第 5 章讨论）或 Run-time Workbench。

Run → *Run*

通过此菜单项可以设置运行配置并运行代码。

讨论

在运行代码之前，保存文件，否则 Eclipse 会提示你保存文件。运行代码的一般方法是选择 Run → Run As，然后从子菜单中能够选择运行代码的方式：Java Applet、Java Application、JUnit Test 或 Run-time Workbench（用于运行开发中的插件）。

注意： Run → Run History 和 Run → Run Last Launched 菜单项提供了运行最近运行过的程序的快捷方式。

Eclipse 3.0

Eclipse 3.0 在 Run → Run As 子菜单中增加了一个菜单项：JUnit Plug-In Test。Eclipse 3.0 使 JUnit 可用于插件开发。

参考

第5章，测试与调试；*Eclipse* (O'Reilly) 一书的第2章。

4.10 编译代码

问题

你想创建编译的 *.class* 文件。

解决方案

默认情况下，在保存代码时将创建或更新编译的 *.class* 文件。也可以手工执行编译。

讨论

默认情况下，当保存一个资源时，将自动创建 *.class* 文件。如果此功能没有打开，可选择 Window → Preferences → Workbench，然后选中 Perform build automatically on resource modification 复选框，如图4-15所示。要关闭自动编译功能，可取消此复选框。

要手工编译项目，可选择 Project → Build Project。

注意： 如果 *.class* 文件已经移动或毁坏，自动编译功能将不起作用，必须通过手工重新编译 *.class* 文件。在这种情况下，可以选择 Project → Build Project。

改变编译顺序

实际上，Eclipse 自身已经通过解释项目引用，设置好了项目的编译顺序。然而，你也可以自己设置编译顺序。选择 Window → Preferences → Build Order，如图4-16所示。

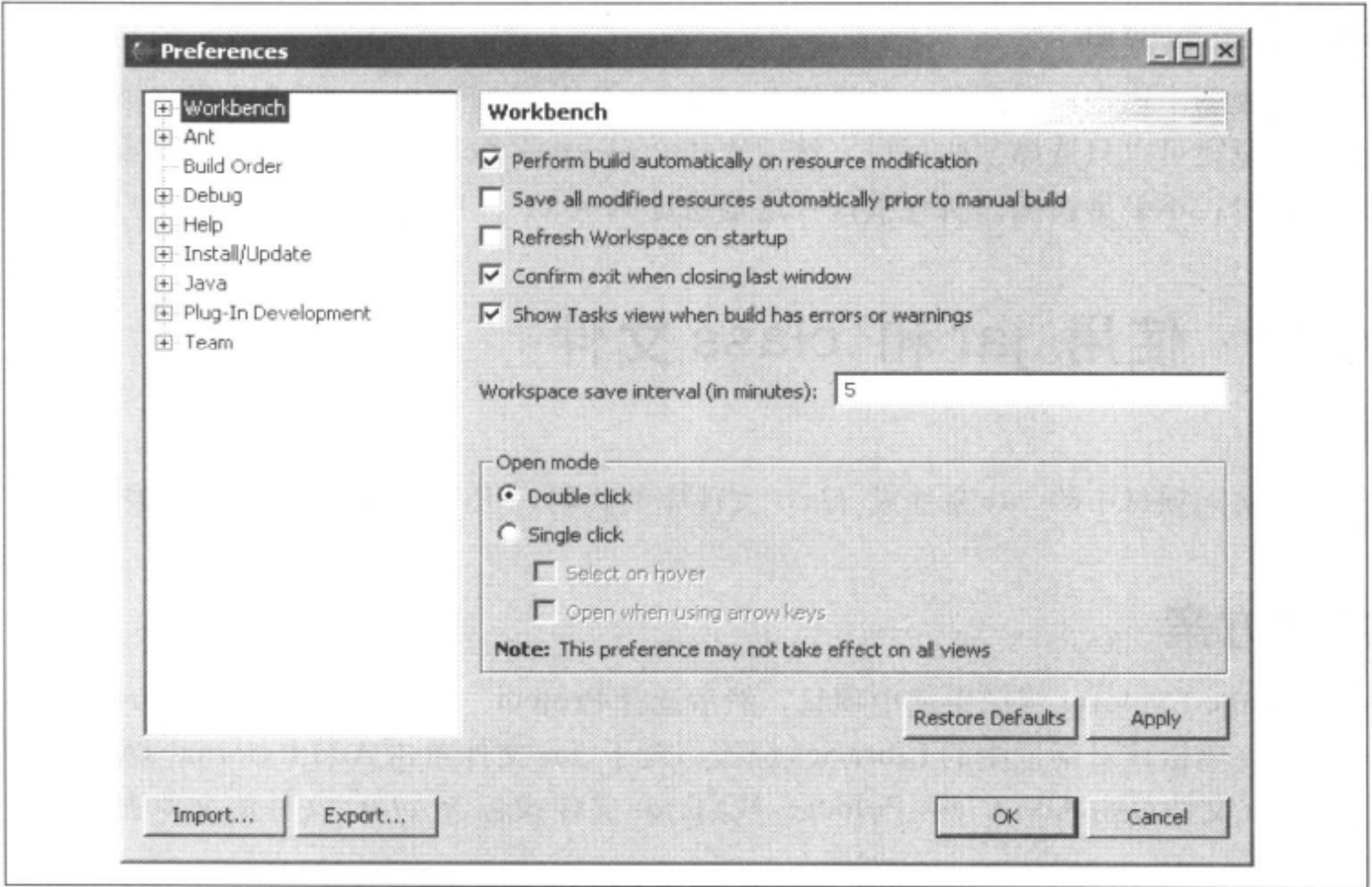


图 4-15：关闭自动编译功能

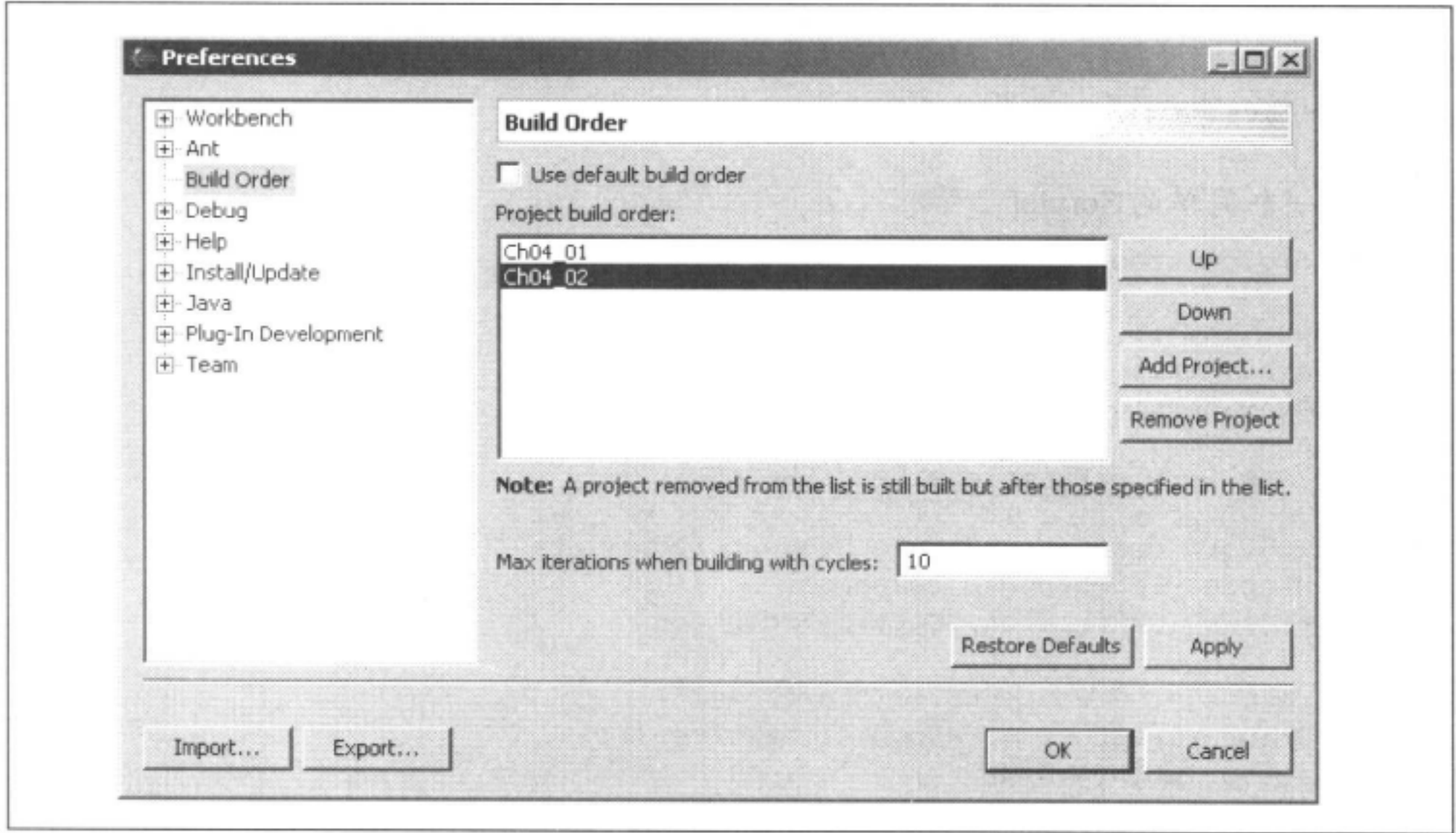


图 4-16：设置编译顺序

编译选定的项目

如果在视图中选中一个项目，并选择 Project → Rebuild Project，所选项目将被重新编译。甚至还可以只选择某些项目，然后选择 Project → Rebuild Project，即可重新编译这些项目，不必重新编译所有项目（这是通过 Project → Rebuild All 来完成的）。

4.11 使用 .jar 和 .class 文件

问题

你需要访问项目中的 .jar 文件或 .class 文件中的代码，但 Eclipse 无法找到这些文件。

解决方案

在 Package Explorer 视图中选中项目，然后选择 Project → Properties，打开 Properties 对话框。单击该对话框中的 Libraries 标签，对于 .jar 文件单击 Add External JARs，对于 .class 文件单击 Add Class Folder，找到 .jar 文件或包含 .class 文件的文件夹，然后单击 OK。

讨论

通常，你需要编译路径（如 .class 文件或 .jar 文件）中的其他代码。例如，假如你正在开发一个 Java Servlet，如例 4-3 所示。

例 4-3：一个简单的 Servlet

```
package org.cookbook.ch04;

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class ServletExample extends HttpServlet {

    public void doGet(HttpServletRequest request,
        HttpServletResponse response)
        throws IOException, ServletException
    {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<HTML>");
        out.println("<HEAD>");
        out.println("<TITLE>");
        out.println("Using Servlets");
        out.println("</TITLE>");
    }
}
```

```
        out.println("</HEAD>");  
        out.println("Using Servlets");  
        out.println("</BODY>");  
        out.println("</HTML>");  
    }  
}
```

对Servlet的大量支持都在 *servlet.jar* 文件中。Eclipse 找不到 *servlet.jar* 文件，当它到达导入语句时，将会有许多红色波浪线出现，如图 4-17 所示。

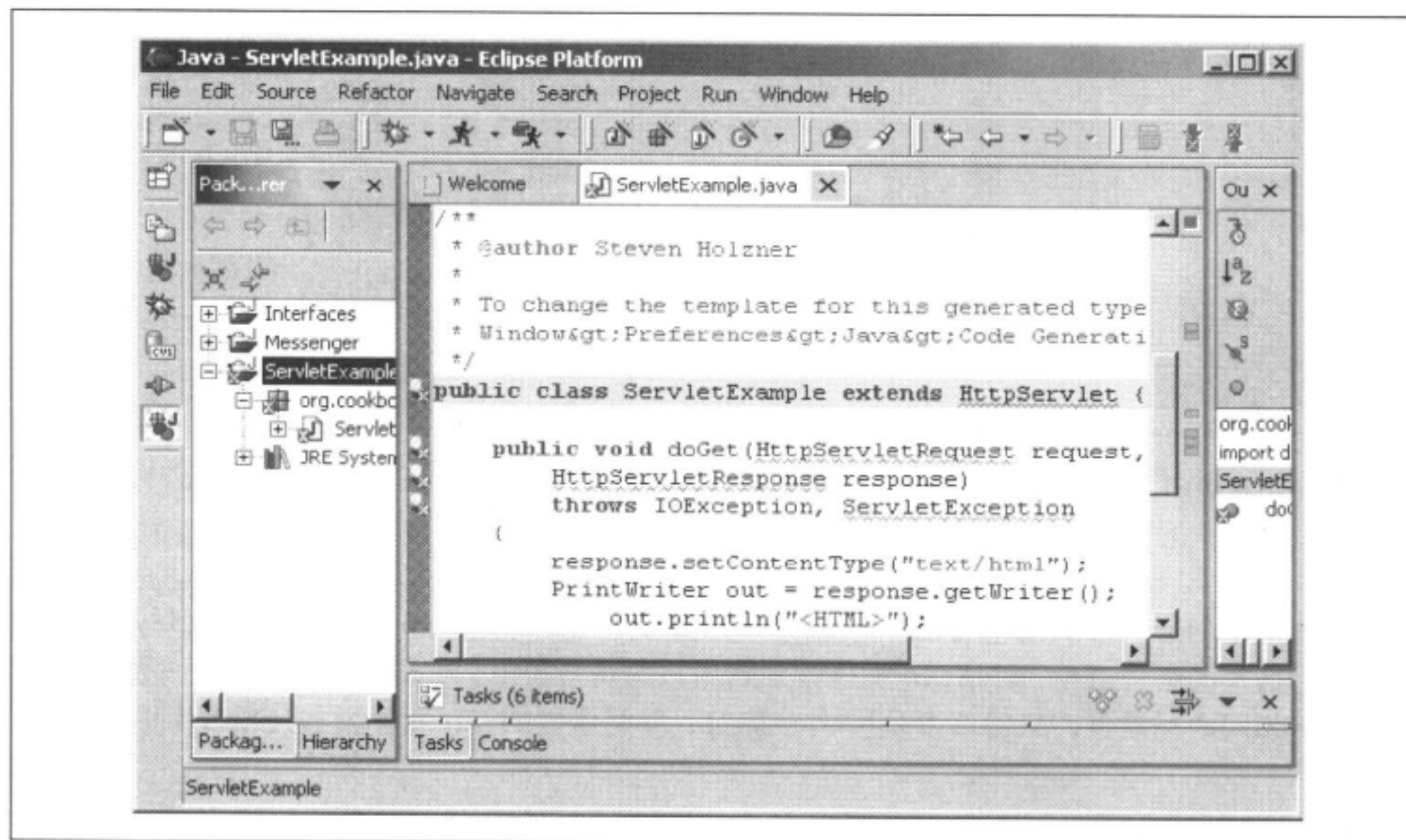


图 4-17：缺失 *servlet.jar* 文件

要将 *servlet.jar* 文件添加到编译路径中，可选择 Project → Properties，并单击 Libraries 标签。然后单击 Add External JARs，找到 *servlet.jar* 文件，并单击 OK。这样做将把 *servlet.jar* 文件添加到编译路径中，如图 4-18 所示。单击 OK，关闭 Properties 对话框，然后编译项目；在执行这些操作时，将会出现好的结果（可以看到 *servlet.jar* 文件出现在 Package Explorer 路径中）。

如果在类路径（classpath）中添加多个 *.jar* 文件，还可以指明搜索这些文件的顺序。只需单击 Properties 对话框中的 Order and Export 标签，并使用 Up 和 Down 按钮改变导入项目的顺序即可。

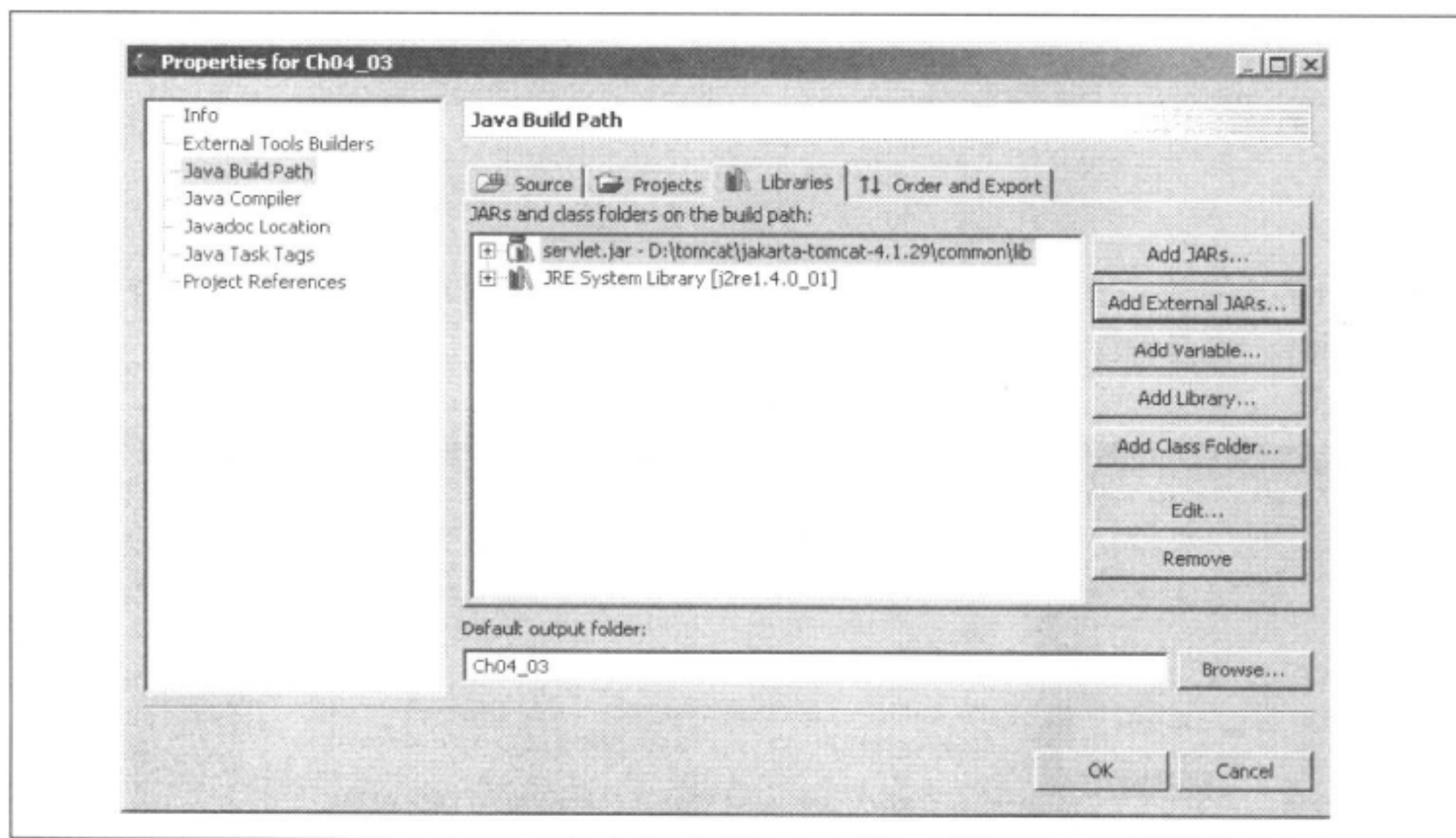


图 4-18: servlet.jar 文件出现在编译路径中

注意： 如果你知道在首次创建项目时要使用一个.jar文件(如servlet.jar文件),可以在New Project对话框的第三页中,将该.jar文件添加到项目的类路径中。在操作时可以看到与图4-18所示相同的标签页。只需单击 Libraries 标签,并将所需的.jar文件添加到项目中即可。

创建类路径变量

如果你知道将要使用一个.jar文件(如servlet.jar文件),可能需要创建一个类路径变量。当需要在项目的编译路径中添加条目时,这样做可以节省时间。这样使用类路径变量不仅是方便的,而且可以集中类路径引用,以便处理。例如,如果需要在多个项目中使用servlet.jar文件的新版本,所需要做的就是更新一个类路径变量。

要创建类路径变量,可选择 Window → Preferences → Java → Classpath Variables,如图4-19所示。单击 New,输入新变量的名称——这里使用 SERVLET_JAR,输入其路径(或浏览到其存储位置),然后单击 OK。在图4-19中可以看到这个新建的变量。

使用类路径变量

如果要将这个类路径变量添加到一个项目的类路径中,可打开项目的 Properties 对话框,单击 Libraries 标签,单击 Add Variable 按钮(如图4-18所示),然后选择要添加到类路径中的变量。

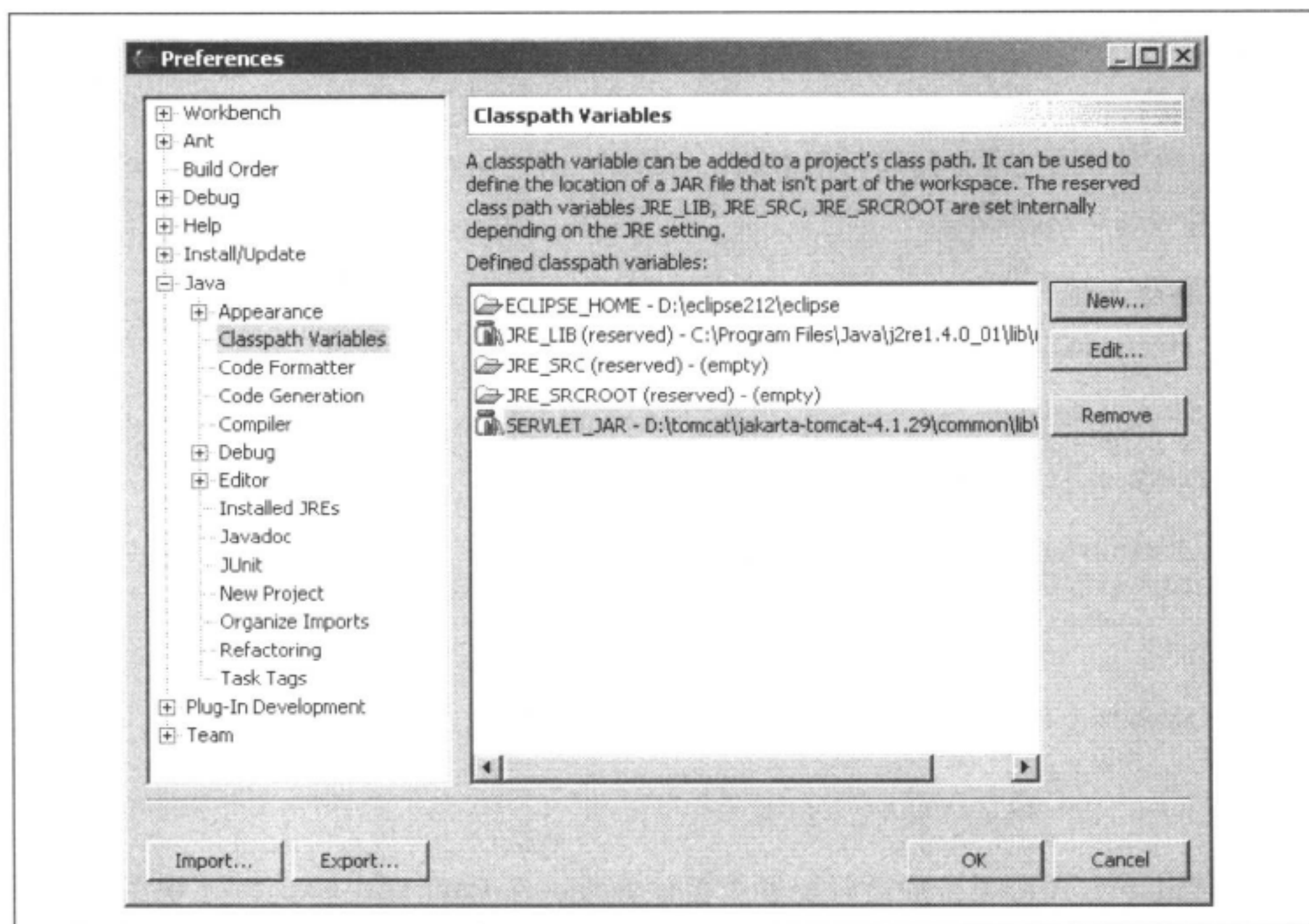


图 4-19：创建类路径变量

参考

1.5 节，创建一个 Java 项目。

4.12 设置运行配置

问题

你想把命令行参数传递到代码中，或者仅为当前项目选择一个 JRE，或者把参数传递给 JVM。

解决方案

通过选择 Run → Run，设置项目的运行配置。

讨论

假设你想运行例4-4中的代码, *Launcher.java*, 它读取命令行参数, 拼接它们, 并显示结果。如果不向该代码中传递任何参数, 将得到一个 `java.lang.ArrayIndexOutOfBoundsException` 异常。

例 4-4: 拼接示例

```
package org.cookbook.ch04;

public class Launcher
{
    public static void main(String[] args)
    {
        String text = "";
        for (int loopIndex = 0; loopIndex < args.length; loopIndex++){
            text += args[loopIndex] + " ";
        }

        System.out.println(text);
    }
}
```

可以通过以下方式在项目的运行配置中提供参数: 在 Package Explorer 视图中选中 *Launcher.java*, 选择 Run → Run 以打开 Run 对话框, 并单击 New 按钮, 为项目创建一个新的运行配置, 如图4-20所示。通过该对话框中的 Main 标签页可以选择项目中的 main 类。通过 Arguments 标签页可以指定命令行参数和 JVM 参数。通过 JRE 标签页可以选择项目的 JRE。而通过 Source 标签页可以将原代码连接到项目中, 以便进行调试。

在 Program arguments 文本框中输入单词 “No problem.”, 如图4-20所示。然后单击 Run, 代码将读取该命令行参数, 并显示它, 如图4-21所示。

参考

4.8 节, 选择 Java 编译运行环境; 4.9 节, 运行代码。

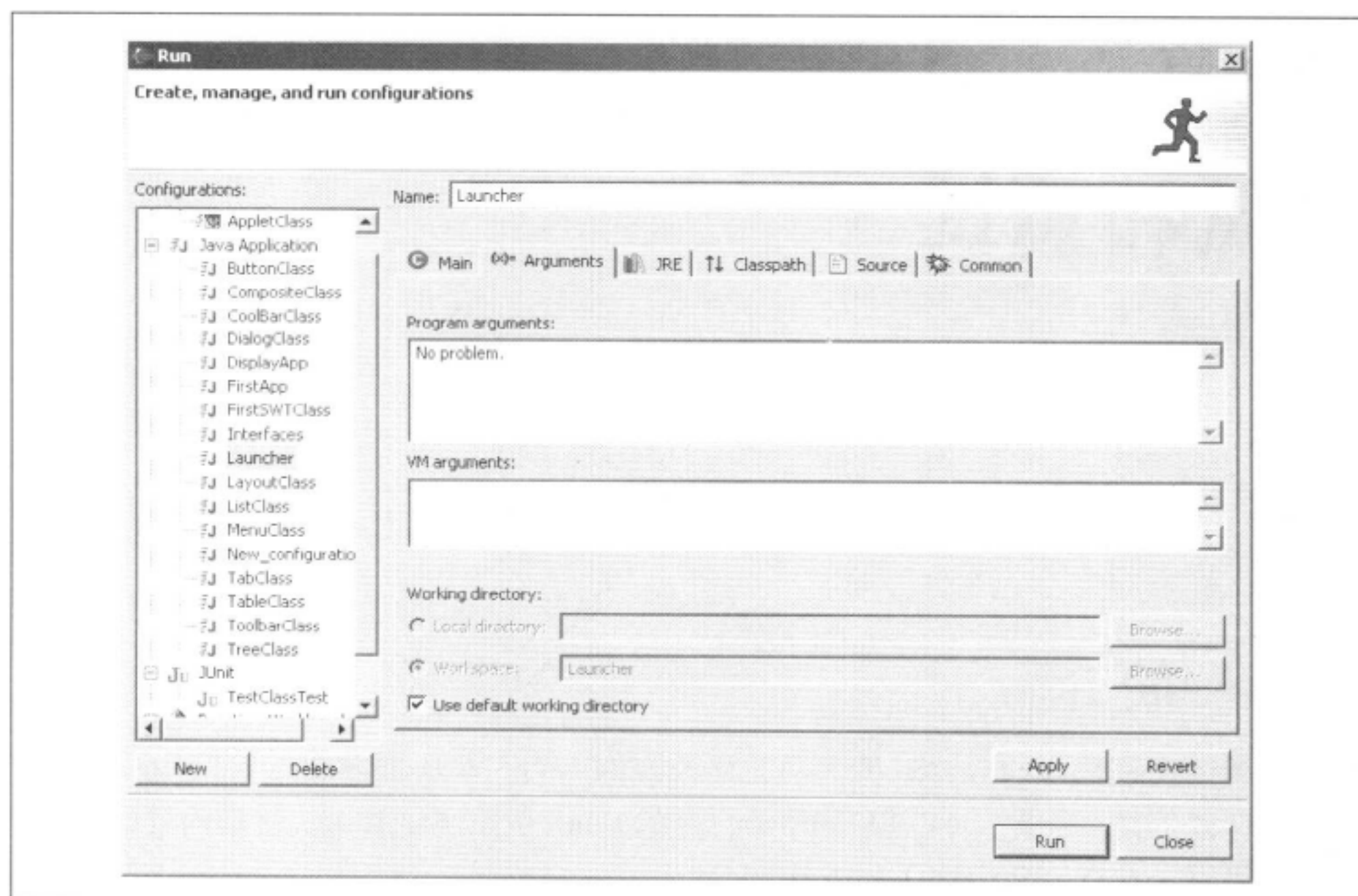


图 4-20：设置运行配置

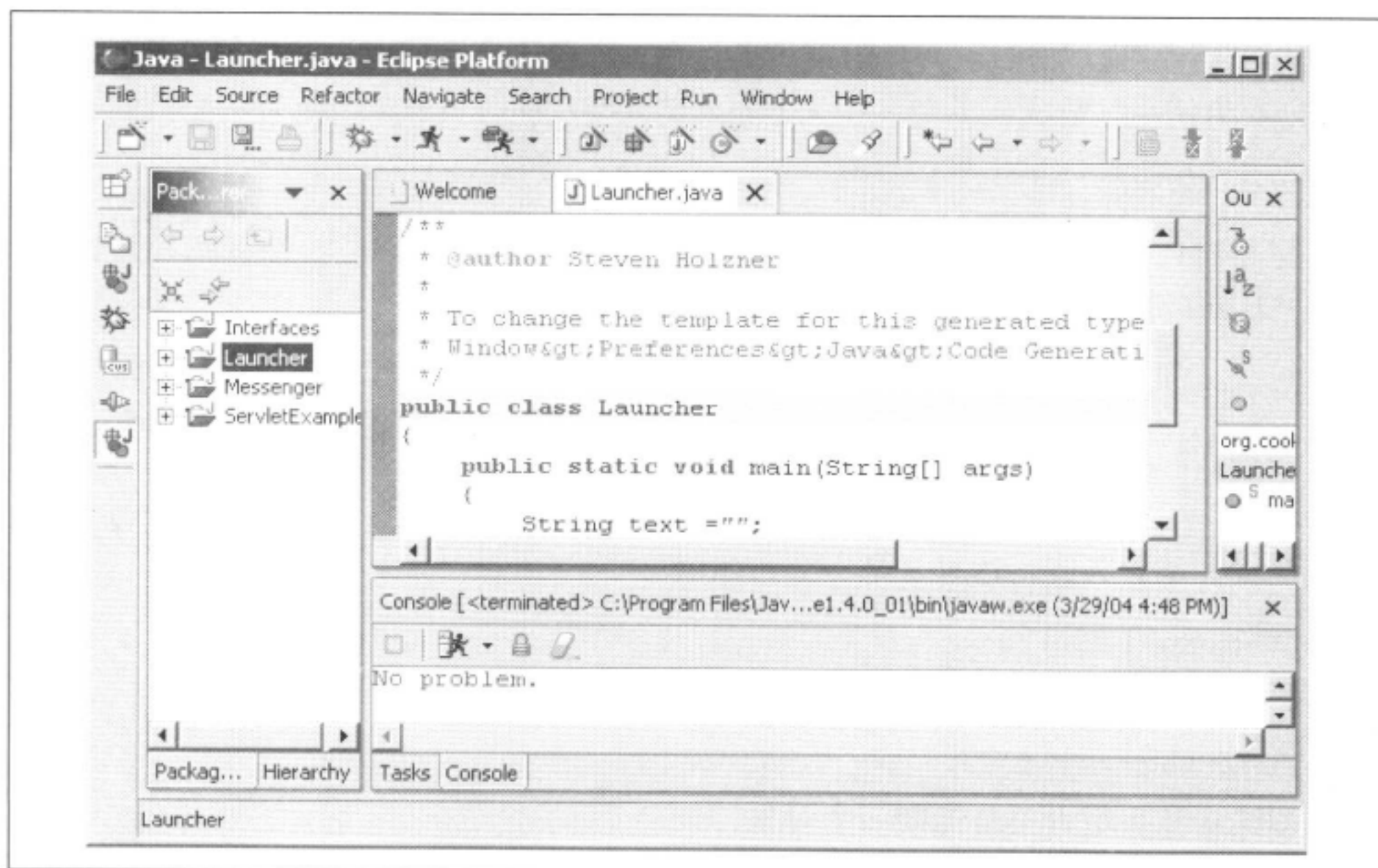


图 4-21：传递命令行参数

第 5 章

测试和调试

5.0 简介

测试和调试代码是 Java 程序开发人员无法改变的事实。本章将重点介绍 Eclipse 自带的测试和调试工具。通常，可以用 Eclipse 自带的 JUnit 包来测试代码，使用内置的调试器来调试代码。本章将介绍这两种工具。

JUnit 是一个开源测试框架（关于开源倡议的进一步信息，请访问站点 <http://www.opensource.org>）。它可以用来测试调用代码中的方法的基于 JUnit 的类的结果。一种 Eclipse 插件可以加快运行 JUnit 测试的过程。使用 JUnit，可以创建标准的测试集，并将其分发给使用你的代码的每一个人；如果某人要编辑你的代码，他所需要做的全部工作就是通过几次鼠标单击来运行测试集，以确保没有对代码产生任何破坏。

JUnit 运行时使用一个断言方法集来调用你的代码，这些方法可测试代码的返回值。下面是一些 JUnit 测试：

`assertEquals(a, b)`

测试 `a` 是否等于 `b`（`a` 和 `b` 是初始值，或者必须有一个 `equals` 方法，用于比较目的）。

`assertFalse(a)`

测试 `a` 是否为假，这里 `a` 是一个布尔值。

`assertNotNull(a)`

测试 `a` 是否不是空值，这里 `a` 可以是一个对象或空值。

`assertNotSame(a, b)`

测试 `a` 和 `b` 是否不是指同一个对象。

`assertNull(a)`

测试 `a` 是否为空值，这里 `a` 可以是一个对象或空值。

`assertSame(a, b)`

测试 `a` 和 `b` 是否指同一个对象。

`assertTrue(a)`

测试 `a` 是否为真，这里 `a` 是一个布尔值。

当在 Eclipse 中运行一个使用 JUnit 插件的 JUnit 应用程序时，应用程序将打开其自己的视图，以立即反馈测试是通过了还是失败了。

5.1 安装 JUnit

问题

你想安装 JUnit，以便在 Eclipse 中使用它。

解决方案

将 *junit.jar* 添加到项目的类路径中。

讨论

在 Eclipse 2.1.x 和最新版本 Eclipse 3.0 中，可以在 *eclipse/plugins/org.junit_3.8.1/junit.jar* 目录下找到 *junit.jar* 文件。

如果你打算经常使用 JUnit，新建一个类路径变量比较方便；本书的类路径变量命名为 `JUNIT`。选择 `Window → Preferences → Java → Classpath Variables`，单击 `New` 按钮，以打开 `New Variable Entry` 对话框；现在输入新变量的名称 `JUNIT` 和至 *junit.jar* 的路径。

还可以让 Eclipse 知道 JUnit 的源代码位于何处；这一步骤对于调试是有益的，可以使 Eclipse 访问 JAR 文件的代码，以便在必要时显示它。注意，这一步骤是可选的，因为你不必看到 JUnit 代码。要为 JUnit 源代码创建一个新的变量（本书称之为 `JUNIT_SRC`），可将该变量连接至 *eclipse/plugins/org.eclipse.jdt.source_x.y.z/src/org.junit_3.8.1/junitsrc.zip*，其中 *x.y.z* 是你的 Eclipse 版本号。然后右击项目，单击 `Properties`，单击 `Java Build Path` 节点和 `Libraries` 标签，然后展开 `JUNIT` 条目的节点，如图 5-1 所示。

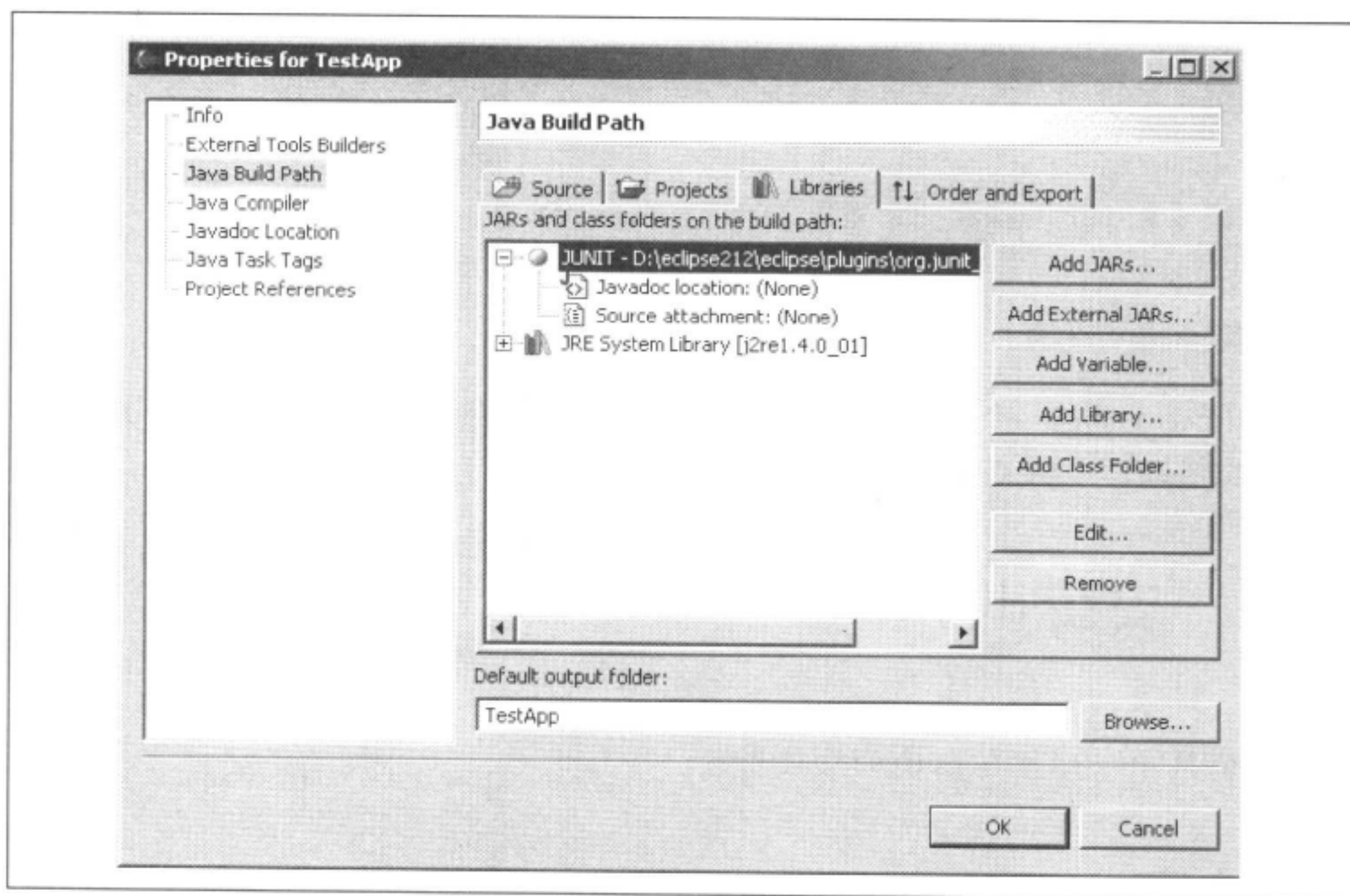


图 5-1：使 Eclipse 能够访问源代码

展开一个 JAR 文件的节点后，可以指定相关源代码和 Javadoc 的位置。要使用源代码的 JUNIT_SRC 变量，可在展开的节点中选择 Source attachment，并单击 Edit，打开 Source Attachment Configuration 对话框。单击 Variable 按钮，双击 JUNIT_SRC 变量，然后单击 OK，关闭该对话框。在图 5-2 中可以看到结果，在这里我们使 JUnit 的源代码可以为 Eclipse 所访问。单击 OK，关闭 Properties 对话框。

注意：这就是将 *junit.jar* 文件添加到编译路径的方法。但结果证明，如果不这样做，也不会损失什么。一旦你打算构建 JUnit 测试用例，JUnit 插件将询问是否要将 *junit.jar* 文件自动添加到编译路径中，如果尚未添加的话。在本例中，该插件将为你完成一切。

参考

4.11 节，使用 *.jar* 和 *.class* 文件；5.2 节，用 JUnit 测试应用程序；*Java Extreme Programming Cookbook* (O'Reilly) 一书中关于 JUnit 的部分内容。

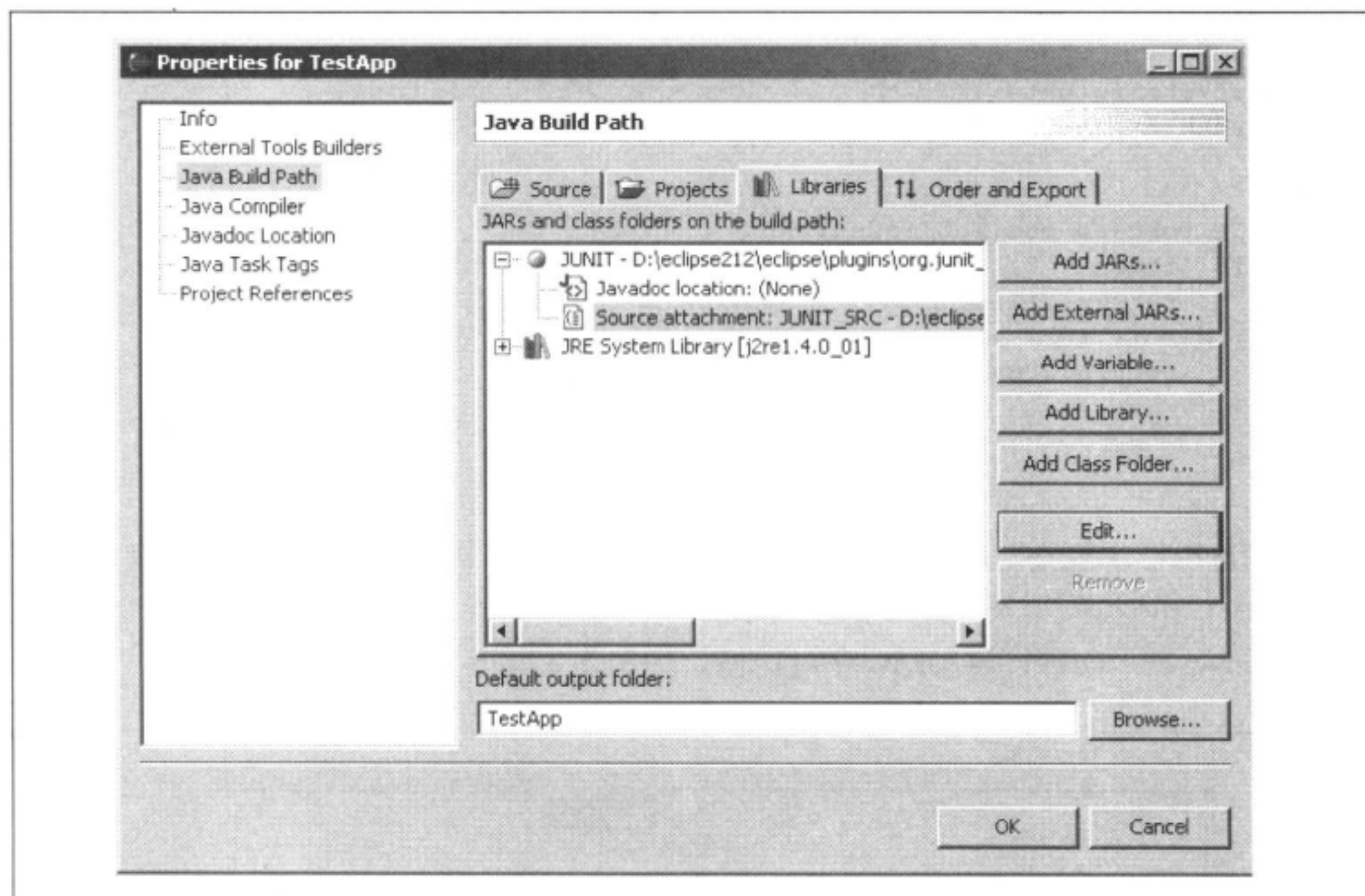


图 5-2：将源代码连接到 .jar 文件

5.2 用 JUnit 测试应用程序

问题

你想创建一个 JUnit 测试用例。

解决方案

创建一个基于 JUnit 的类，并实现要运行的测试。然后立即使用 JUnit 插件查看测试结果。

讨论

作为一个例子，我们将测试应用程序 *TestApp*，如例 5-1 所示，其中使用了一个名为 *TestClass* 的类。该应用程序有两个方法：`get` 和 `set`，前者返回一个字符串，而后者在传递的值等于或大于 0 时返回 `true` 的确认值。

例 5-1：一个简单的 Java 类

```
package org.cookbook.ch05;
```



```
public class TestClass
{
    public String get() {
        return "Test String";
    }

    public boolean set(int index) {
        if (index < 0) {
            return false;
        } else {
            return true;
        }
    }
}
```

为了测试这个应用程序，使用JUnit Wizard插件在项目中创建一个新的类，新建类是对JUnit TestCase类的扩展。要调用这个向导，可右击要测试的类，本例中是TestClass，然后选择New → Other，打开New对话框，如图5-3所示。

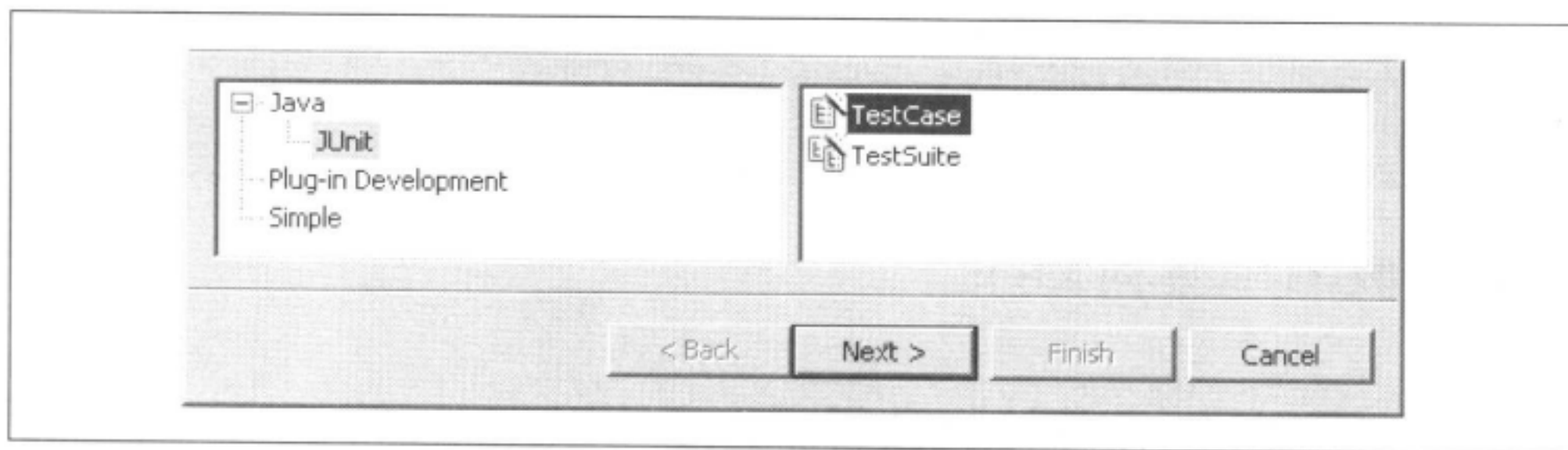


图 5-3：新建一个基于 JUnit TestCase 的类

在左边的列表框中展开 Java 节点，并选择 JUnit。在右边的列表框中，选择 TestCase。单击 Next，显示如图 5-4 所示的对话框。

JUnit 测试用例的命名约定是在要测试的类名称前加上“Test”。在本例中，JUnit 建议了一个测试用例名称 TestClassTest。一定要选中 setUp() 和 tearDown() 复选框，如图 5-4 所示。使用这两个方法可以在测试用例中建立、用后清除数据和/或对象（这两个方法的 JUnit 术语是夹具）。单击 Next，打开下一个对话框，如图 5-5 所示。

在这个对话框中，选择要测试的方法，以便 JUnit Wizard 能够为这些方法创建占位程序。在本例中，我们要测试 set 方法和 get 方法，所以选中这两个方法，如图 5-5 所示，并单击 Finish。

这将创建 TestClassTest 类，如例 5-2 所示。可以看到 TestClassTest 类中用于测试 set 方法和 get 方法的占位程序；这两个占位程序分别命名为 testSet 和 testGet。

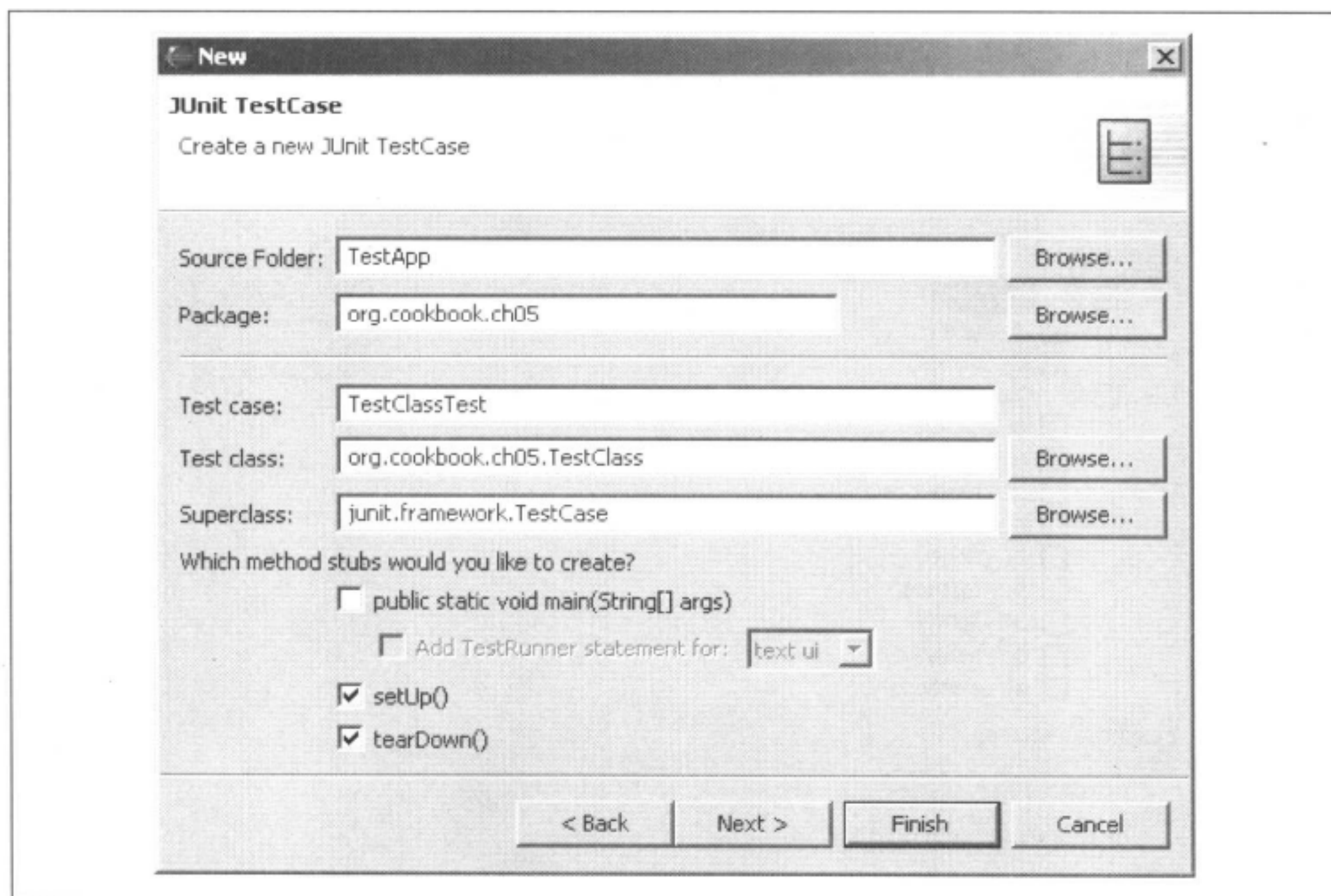


图 5-4：配置 JUnit 测试用例

例 5-2：测试 TestClass

```
package org.cookbook.ch05;

import junit.framework.TestCase;

public class TestClassTest extends TestCase
{
    /**
     * Constructor for TestClassTest.
     * @param arg0
     */
    public TestClassTest(String arg0)
    {
        super(arg0);
    }

    /**
     * @see TestCase#setUp()
     */
    protected void setUp() throws Exception
    {
        super.setUp();
    }
}
```

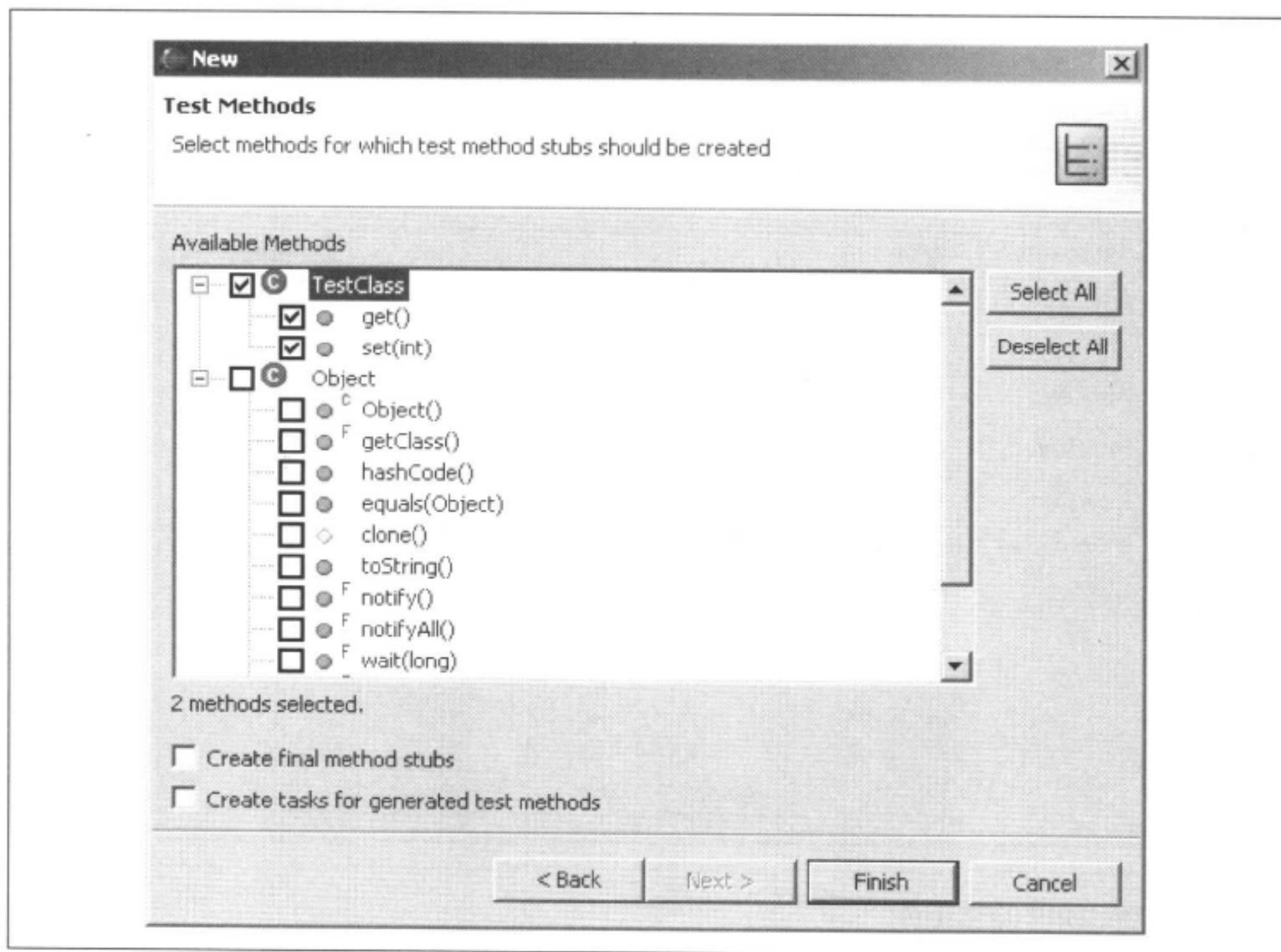


图 5-5: 选择要用 JUnit 测试的方法

```

/*
 * @see TestCase#tearDown()
 */
protected void tearDown() throws Exception
{
    super.tearDown();
}

public void testGet()
{
}

public void testSet()
{
}
}

```

要测试代码，可以将代码添加到调用待测试方法的 JUnit 占位程序中。要在代码中调用非静态的 `get` 方法和 `set` 方法，需要 `TestClass` 类的一个对象，本例中将其命名为 `testClassObject`。在 JUnit `setUp` 方法（每个 JUnit 测试开始前调用该方法）中创建该对象：

```
public class TestClassTest extends TestCase
{
    TestClass testClassObject;
    .
    .
    .
    protected void setUp() throws Exception
    {
        super.setUp();
        testClassObject = new TestClass();
    }
}
```

在创建了 testClassObject 对象之后, 就可以进行测试了。在 testGet 方法中, 可以测试 TestClass 类的 get 方法, 它应该返回一个 String 对象。下面是检查该返回值的方法:

```
public void testGet() {
    assertEquals(testClassObject.get(), "Test String");
}
```

set 方法如果成功的话应该返回 true, 下面用一个 assertTrue 调用来测试 set 方法:

```
public void testSet() {
    assertTrue(testClassObject.set(-1));
}
```

在添加了上述测试代码之后, 在 Package Explorer 视图中选中 TestClassTest 类, 并选择 Run → Run As → JUnit Test, 打开图 5-6 左侧所示的视图。

该视图顶部的红色栏 (本图中显示为黑白色) 表明存在一个错误, 而且如果看一下 JUnit 视图中的 Failures 标签下方, 可以发现 testSet 测试失败了。这是因为, 如果给 set 方法传递一个非负值, 它只返回 true, 但我们给它传递了值 -1, 这意味着基于返回值运行的 assertTrue 调用将失败:

```
public void testSet() {
    assertTrue(testClassObject.set(-1));
}
```

Change that code by passing set a value of 1 instead:

```
public void testSet() {
    assertTrue(testClassObject.set(1));
}
```

现在, 重新运行测试, 将看到表示成功的绿色栏, 如图 5-7 所示 (本图显示为较浅的黑白色)。测试通过了。系统运行成功!

可以看到，这种测试方法是可行的。当你（或任何其他入）对代码进行修改时，只需再次运行测试用例即可，如果一切正常，可以立即看到绿色栏。

Eclipse 3.0

在编写本书之时，JUnit 过程与 Eclipse 3.0 中的过程类似，只是在 3.0 中有更多的选项，比如，可以指定除了 `setUp` 方法和 `tearDown` 方法以外，是否还需要在测试用例中添加一个构造函数。

参考

5.1 节，安装 JUnit; *Java Extreme Programming Cookbook* (O'Reilly) 一书中关于 JUnit 的部分内容; *Eclipse* (O'Reilly) 一书的第 3 章。

5.3 启动调试会话

问题

你编写的代码不能如期运行，代码该调试了。

解决方案

用 Run 菜单中的菜单项启动一个新的调试会话。然后使用各种调试选项，比如单步调试、设置断点等等。

讨论

假设你想让代码显示以下输出：

```
3...
2...
1...
Houston, we have liftoff.
```

首次尝试使用可能与例 5-3 相似的代码，该代码位于一个名为 *DebugApp* 的应用程序中的 *DebugClass* 类中。

例 5-3: DebugApp 示例

```
package org.cookbook.ch05;

public class DebugClass
{
```

```
public static void main(String[] args)
{
    for(int loopIndex = 3; loopIndex > 0; loopIndex--)
    {
        System.out.println(loopIndex + "...");
        if(loopIndex == 0)
        {
            System.out.println("Houston, we have liftoff.");
        }
    }
}
```

遗憾的是，例 5-3 中的代码得到了如下结果：

```
3...
2...
1...
```

代码需要调试。使用 Run 菜单中的下列菜单项之一，可以启动一个调试会话：

Run → Debug History

通过该菜单项可以选择最近运行过的项目，进行调试。

Run → Debug As

通过该菜单项可以选择要从子菜单运行的会话类型（Java Applet、Java Applications、JUnit Test 或 Run-time Workbench）。

Run → Debug

通过该菜单项可以为调试会话设置运行配置。

要启动对 *DebugApp* 代码的调试，可选择 Run → Debug As → Java Application，这将在 Debug 透视图打开并运行代码，如图 5-8 所示。

注意图 5-8 所示的 Debug 透视图。可以看到，所调试的程序条目位于 Debug 视图的左上角。在 Debug 视图中，在单词 Debug 旁边有 5 个按钮。从左至右，5 个按钮的名称分别为 Resume（再次执行代码）、Suspend（暂停执行代码，比如当遇到失控的无限循环时）、Terminate（停止调试）、Disconnect（从调试会话断开）和 Remove All Terminated Launches。

注意：正如图 5-8 所示，Debug 视图显示代码的所有终止运行。这个视图很快就会填满。要清除这些终止的运行，可单击视图工具栏上的 Remove All Terminated Launches 按钮。通常，终止的会话会被保留，以防未来某时你可能想返回到某些会话，对代码当前的运行情况与以前的运行情况进行比较。

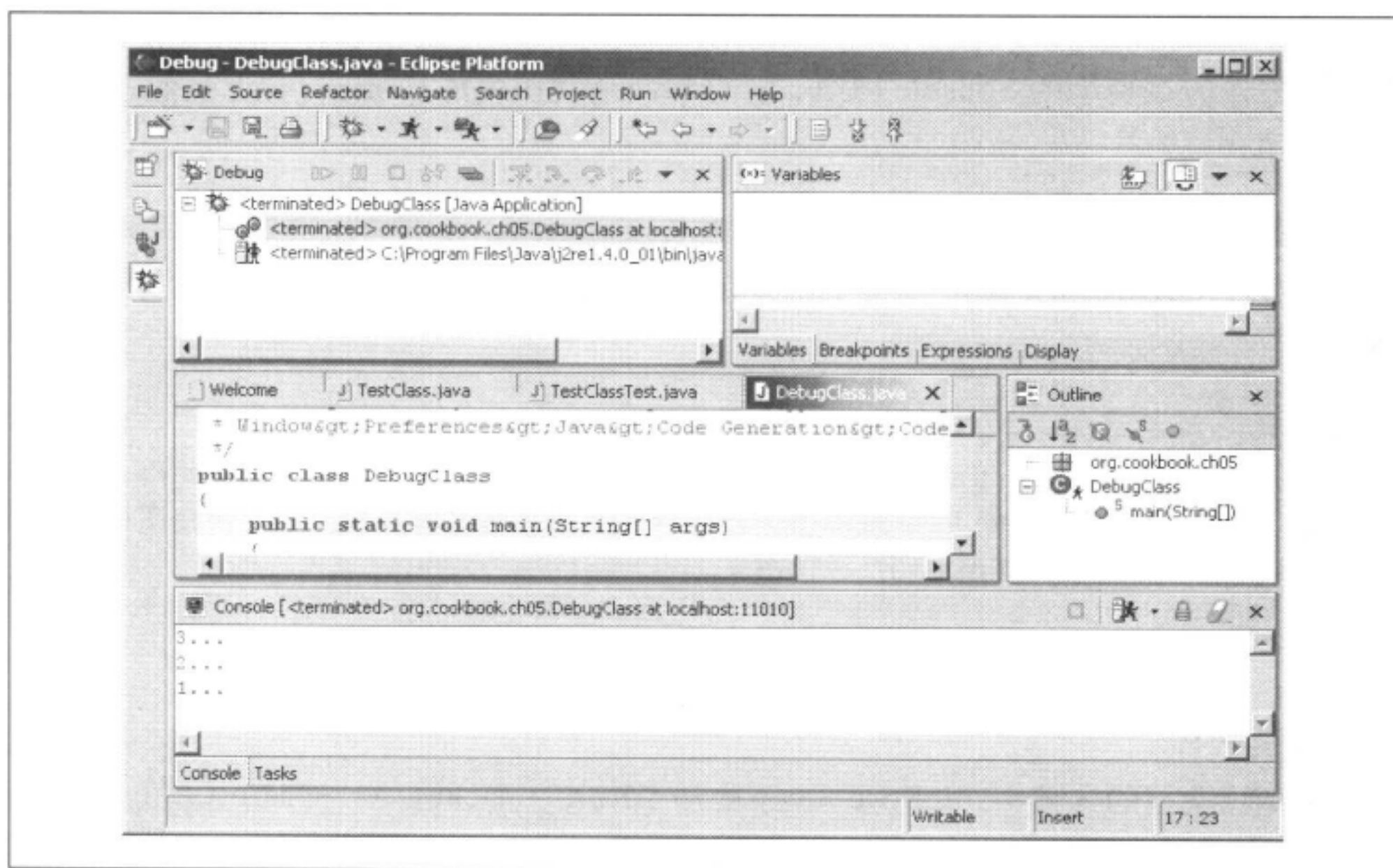


图 5-8：首次尝试调试代码

Debug 视图的右边是一组叠加在一起的视图：Variables 视图、Breakpoints 视图、Expressions 和 Display 视图。

通过 Debug 透视图的 Variables 视图，可以检查局部变量的值。在调试代码时，可以编辑这些值（本章稍后介绍）。在这种情况下，可以与运行的程序进行交互。

注意： 为了表明 Variables 视图中的变量的值已经改变，Eclipse 会把变量的颜色改为红色。

通过 Debug 透视图的 Breakpoints 视图，可以管理代码中的断点，方法是：右击列表中的断点，然后从快捷菜单中选择 Disable、Enable、Remove 或 Remove All。

通过 Debug 透视图的 Expressions 视图，可以求表达式的值（将在 5.12 节介绍）。当在编辑器中右击一个表达式，然后单击 Inspect 时，将计算表达式的值，并在 Expressions 视图中显示计算结果。同样，当单击快捷菜单中的 Display 命令时，结果将出现在 Display 视图中。

Debug 透视图下的编辑器与 JDT 编辑器非常相似；只需要在元素上滚动鼠标，即可查看这些元素的值。

紧邻编辑器的是 Outline 视图，如图 5-8 所示，它与 Java 透视图中的 Outline 视图相同。在编辑器下方是标准的 Console 视图，它与 Java 透视图中的 Console 视图类似，显示程序的输出。

图 5-8 表明，代码已经运行并终止，其结果与以前相同，没有显示预期的“Houston, we have liftoff.”消息。若要在程序运行时查看运行情况，请参阅关于设置断点的 5.4 节。

注意：在结束调试会话时，仍然处于 Debug 透视图。通过选择 Window → Open Perspective → Java，可以切换回 Java 透视图，但在调试期间，通过单击最左边的快捷图标（如图 5-8 所示），可以更方便地切换透视图。

Eclipse 3.0

在启动调试会话的选项中，Eclipse 3.0 还增加了 Run → Debug As → JUnit Plug-in Test 菜单项。默认情况下，Breakpoints 视图将独立显示在 Outline 视图下方，而不是叠加显示。

参考

5.12 节，计算表达式的值；5.4 节，设置断点；5.5 节，单步调试代码；*Eclipse* (O'Reilly) 一书的第 3 章。

5.4 设置断点

问题

你想在代码执行时进行观察，以查明一个问题。

解决方案

设置一个断点，并启动一个调试会话，这将使代码一直执行下去，直至遇到断点。当执行暂停时，可以查看代码执行时发生了什么。

讨论

通过设置断点，可以在程序运行时停止代码的执行，并查看代码执行情况。要在 JDT 编辑器中设置断点，可以双击要设置断点的可执行代码行旁边的标记栏（或者选中代码行，

并选择 Run → Add/Remove Breakpoint, 以设置断点)。以后若要删除断点, 只需再次双击标记栏即可。

为了弄清如何设置断点, 下面将在我们一直在开发的代码中设置一个断点:

```
public class DebugClass
{
    public static void main(String[] args)
    {
        for(int loopIndex = 3; loopIndex > 0; loopIndex--)
        {
            System.out.println(loopIndex + "...");
            if(loopIndex == 0)
            {
                System.out.println("Houston, we have liftoff.");
            }
        }
    }
}
```

断点图标 (一个小圆点) 出现在 JDT 编辑器中这一行代码的左侧, 如图 5-9 所示。

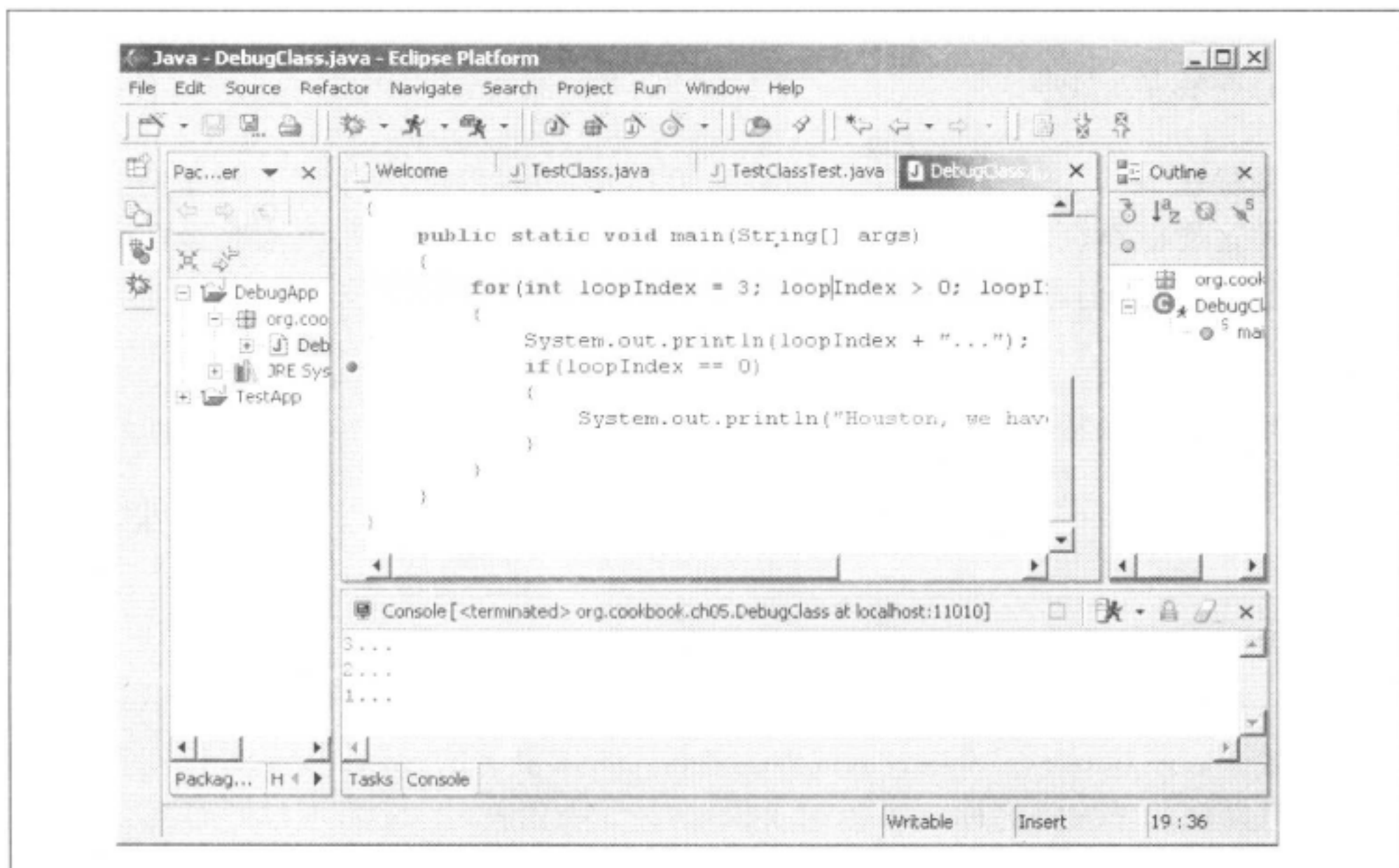


图 5-9: 新增一个断点

现在, 选择 Run → Debug As → Java Application, 你将看到应用程序在断点处停止, 如图 5-10 所示。

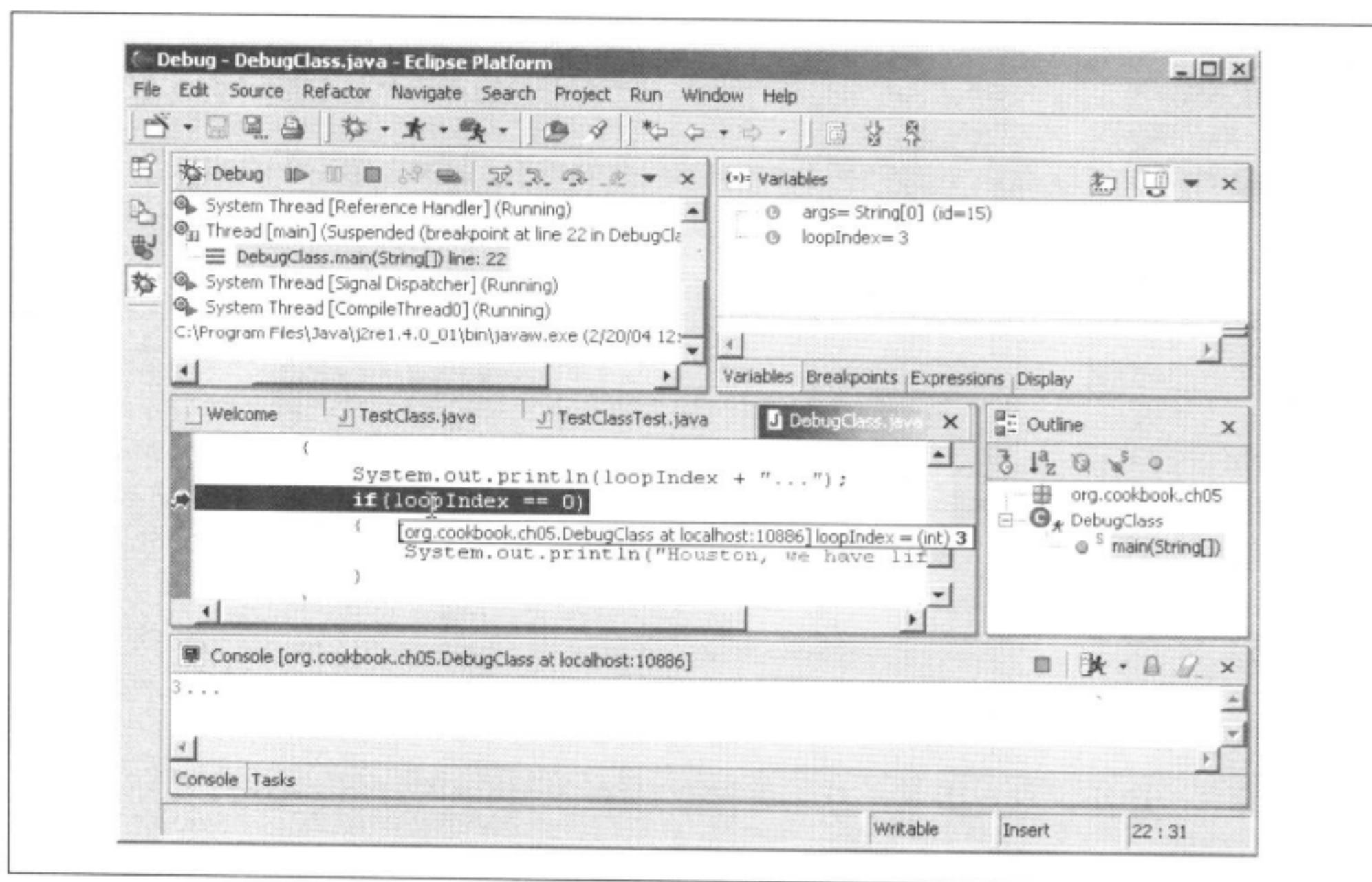


图 5-10：在断点处暂停

将光标停留在 `loopIndex` 变量上，如图 5-10 所示，程序显示该变量存储了值 3，在循环开始时这是正确的。那么，之后该怎么办？可以单步执行代码，查看其运行情况。详细内容请参阅 5.5 节。

注意：要终止调试会话，例如需要在断点处暂停时，可以在 Debug 视图中单击 Terminate 按钮（方形红色按钮），或选择 `Run → Terminate`。

Eclipse 3.0

在 Eclipse 3.0 中，可以在外部源代码中设置断点，也就是说，源代码不在 Java 项目的编译路径上。在 Eclipse 3.0 中，调试器也进行了许多显著的改进；例如，在 Eclipse 3.0 中，调试器使用虚构的名称，如 `arg1`、`arg2` 等，来代表方法参数变量，而真实的名称在编译代码中不可用。这一版本还提供了新的注视，用于在执行调用栈中突出当前指令指针和框架（可以通过选择 `Window → Preferences → Java → Editor → Annotations` 进行配置）。

参考

5.3 节，启动调试会话；5.5 节，单步调试代码；*Eclipse* (O'Reilly) 一书的第 3 章。

5.5 单步调试代码

问题

代码在断点处停止执行，并且你想逐行执行代码，直至遇到下一个断点。

解决方案

使用代码单步执行选项，可以通过工具栏按钮、菜单项或键盘快捷键，来访问这些选项。

讨论

通过暂停的代码的最基本的方法是单步执行。Eclipse 提供了 4 个主要的选项，对应于 Debug 视图工具栏（从双头箭头往右）上的 4 个箭头按钮（在调试会话期间暂停时，通过 Run 菜单也可以访问这些选项）：

Step With Filters (Shift-F5)

使用预定义的或者你创建的过滤器进入选定的语句。如果所进入的语句是一个方法调用，除非已经跳过了所调用方法，否则将在该方法内部继续执行。

Step Into (F5)

进入选定的语句。如果该语句是一个方法调用，将在所调用方法内部继续执行。

Step Over (F6)

越过选定的语句。不进入方法调用。

Step Return (F7)

在到达当前方法结尾之前继续执行，然后返回，从方法返回之后（或遇到一个断点时）暂停。

在本章的例子中，单击 Step Into 按钮，使调试器移动到可执行代码的下一行。由于上一节中在 `if(loopIndex == 0)` 行暂停，且 `loopIndex` 等于 3，而不是 0，所以 `if` 语句体没有执行，而是进入循环的下一次迭代，如图 5-11 所示。

注意：注意，Debug 视图左上角显示了当前在代码中的位置。在本例中，当前正在执行主线程中的代码，而且位于 `DebugClass.main` 栈框架中（栈框架用由 3 个水平条组成的图标标记）。当要调试的代码中包含许多层方法调用时，知道在栈框架中的位置是非常有用的。要从任何一个栈框架中恢复调试，只需在 Debug 视图中选择该栈框架即可。

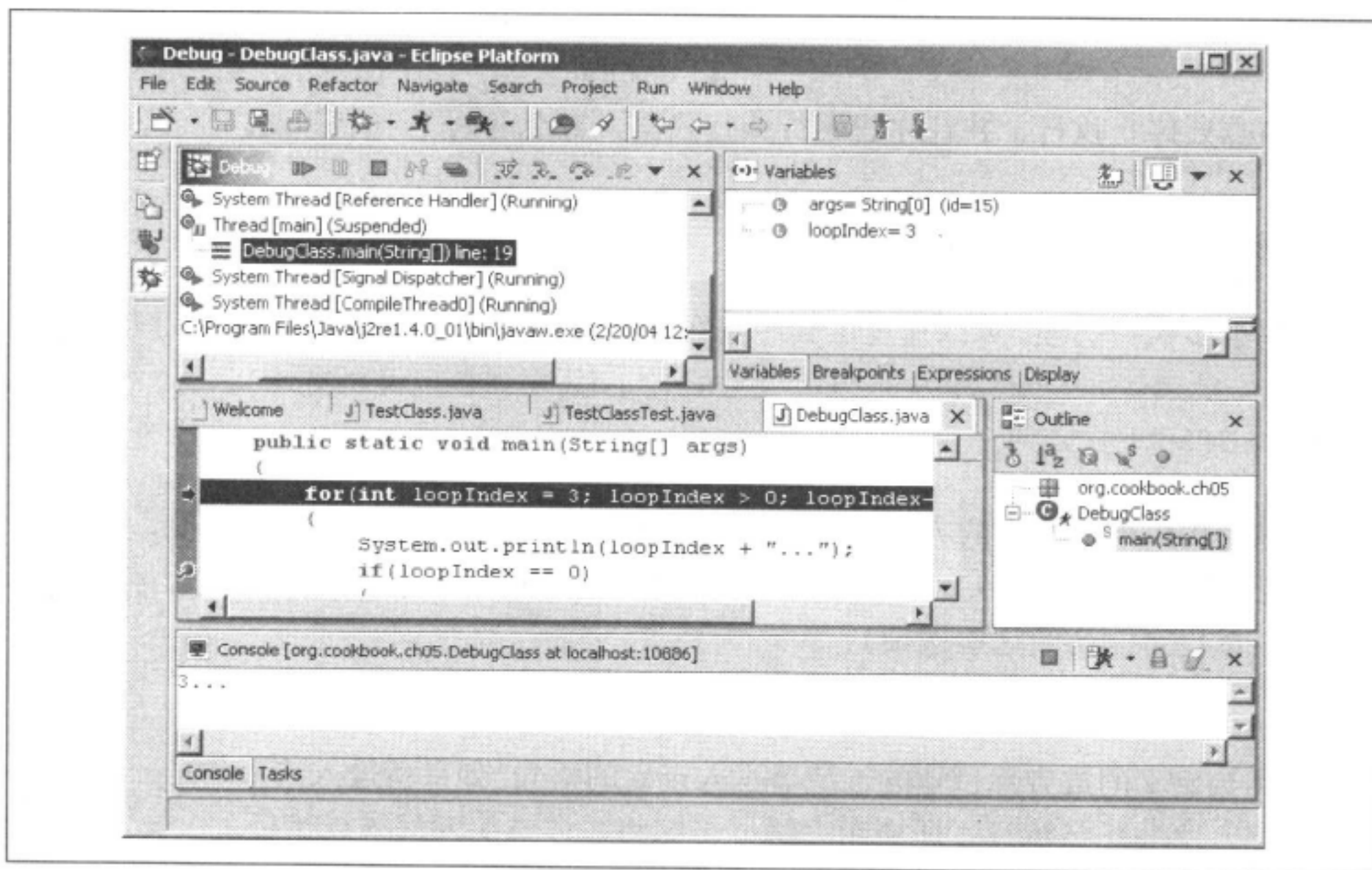


图 5-11：单步调试代码

在单步执行代码时，还可以使用单步过滤器来指定要避开（或跳过）的代码。例如，跳过一个类或包中的代码就意味着，在选择 Step With Filters，不会单步执行代码。

通过选择 Window → Preferences → Java → Debug → Step Filtering，可以设置单步过滤器，如图 5-12 所示。如果需要跳过代码，可选中你在该图中看到的预定义过滤器旁边的复选框，或者通过单击 Add Filter 按钮新建一个过滤器。

当代码不多时，逐行单步执行代码是没有问题的。但是，如果调试的是具有数百行的代码，就需要某种更快的方法。在这种情况下，请参阅 5.6 节，在遇到断点前持续运行。

参考

5.3 节，启动调试会话；5.4 节，设置断点；5.6 节，在遇到断点前继续运行；*Eclipse* (O'Reilly) 一书的第 3 章。

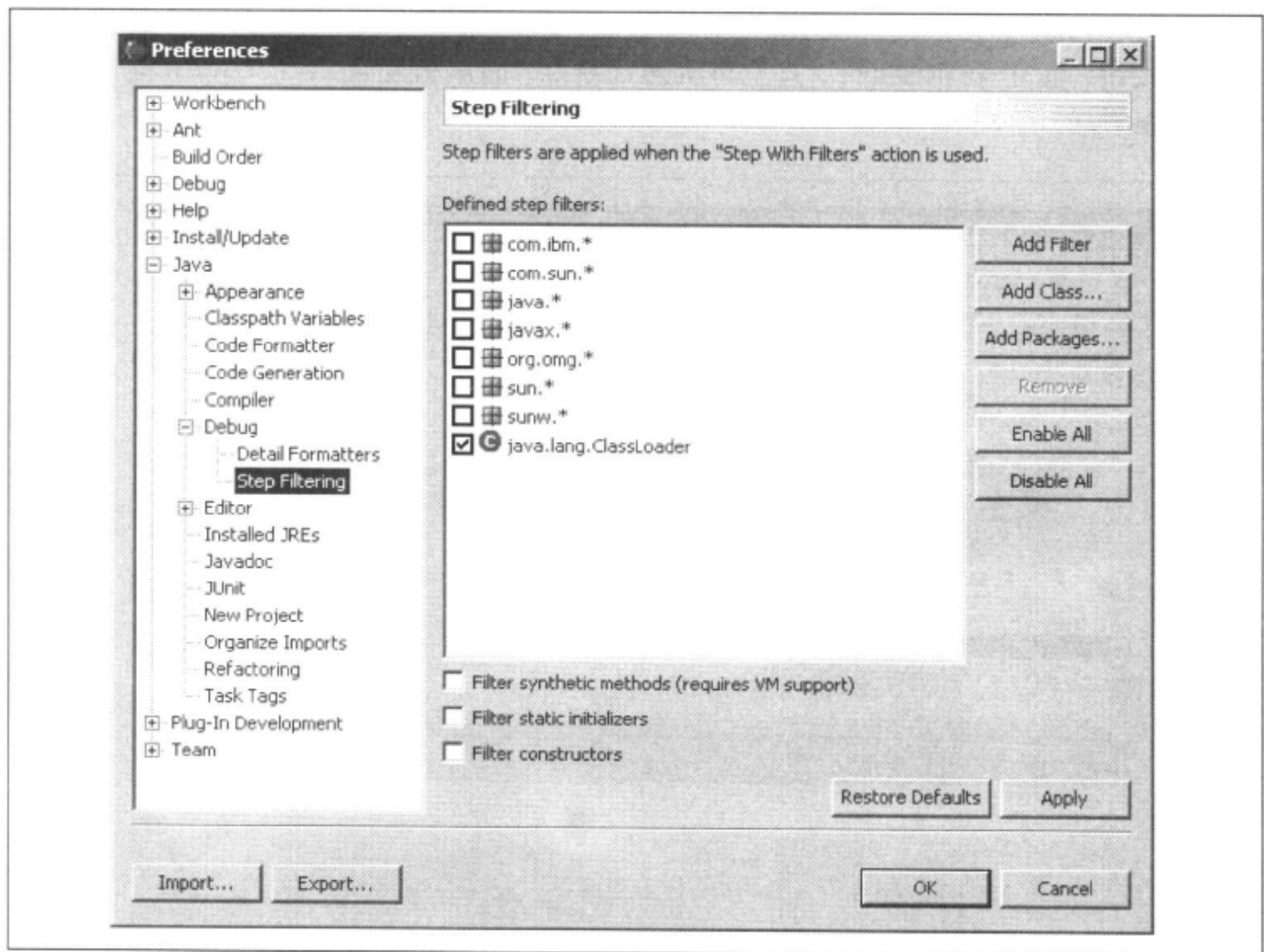


图 5-12：指定单步过滤器

5.6 在遇到断点前继续运行问题

你不想在调试时单步调试许多行代码，而只想单步调试断点。

解决方案

当在断点处暂停执行时，在 Debug 视图中单击 Resume 按钮，或者选择 Run → Resume。执行将继续，直至遇到下一个断点。

讨论

如果不想单步调试代码，还有其他的选择。例如，可以简单地让代码执行下去，直至到达一个断点。为此，只需在 Debug 视图中单击 Resume 按钮（Debug 视图中单词 Debug 右边的箭头按钮），或选择 Run → Resume。

在上一节中，程序在一个断点处停下来，然后单步执行到可执行代码的下一行。单击 Resume 按钮，恢复程序的执行，直至再次遇到断点，而索引变量 `loopIndex` 将等于 2，如图 5-13 所示。

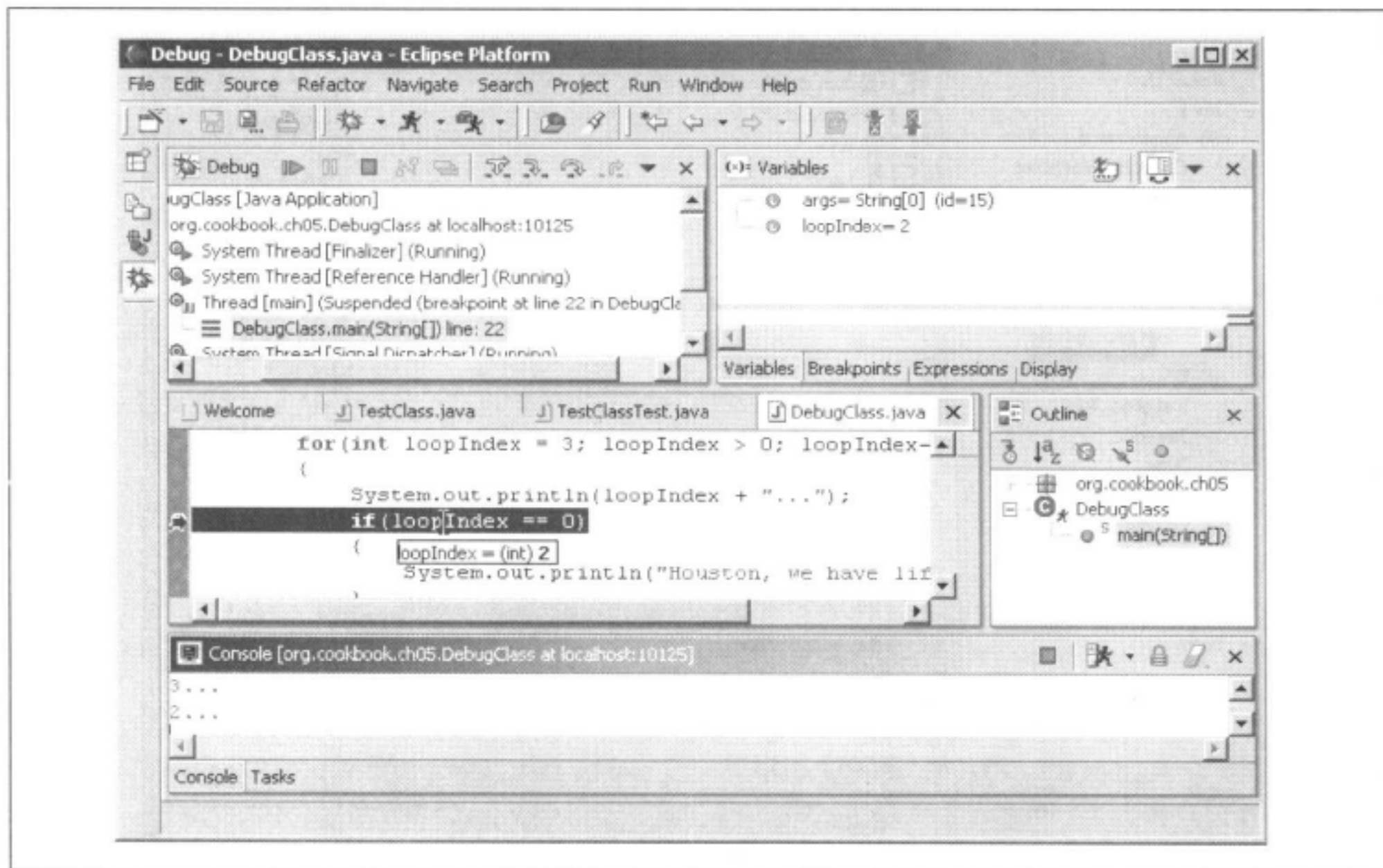


图 5-13：运行至一个断点

再次单击 Resume 后，将进入下一次迭代，而 `loopIndex` 将等于 1。再次单击 `loopIndex`，使代码继续执行，直至程序结束，这有点出乎意料，因为我们期待着 `loopIndex` 等于 0。得到一个意外的结果，表明代码中存在错误 (bug)；问题出在错误地建立循环的那一行：

```
public class DebugClass
{
    public static void main(String[] args)
    {
        for(int loopIndex = 3; loopIndex > 0; loopIndex--)
        {
            System.out.println(loopIndex + "...");
            if(loopIndex == 0)
            {
                System.out.println("Houston, we have liftoff.");
            }
        }
    }
}
```


for 循环应该这样编写：

```
public class DebugClass
{
    public static void main(String[] args)
    {
        for(int loopIndex = 3; loopIndex >= 0; loopIndex--)
        {
            System.out.println(loopIndex + "...");
            if(loopIndex == 0)
            {
                System.out.println("Houston, we have liftoff.");
            }
        }
    }
}
```

问题这样就解决了，如图 5-14 所示，而经过调试的代码将如期运行。

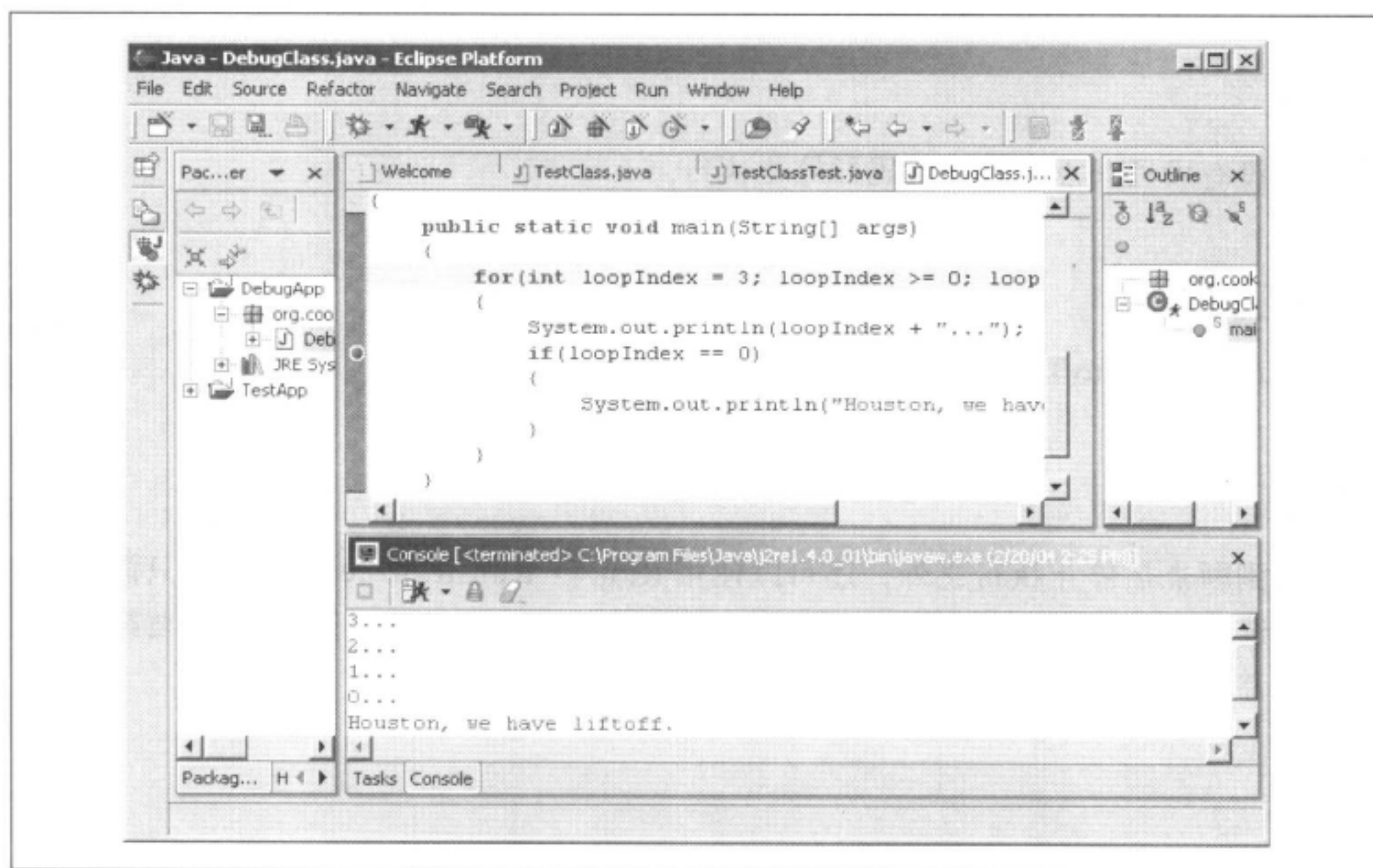


图 5-14：可行的代码

除了恢复执行直至遇到断点，还可以选择 Run → Run to Line。只需单击一行，并选择该菜单项，代码将继续执行到该行。当调试要跳过大段的代码并快速设置特别的断点时，这种功能是非常有用的。

还有另一种选择。通过选择 Run → Step Into Selection，可以指明要进入的方法。在当前执行的语句行中，把光标放置在要进入的方法名称上，并单击 Step into Selection。

Eclipse 3.0

在 Eclipse 3.0 中，Step into Selection 命令不再限于当前执行的行（在 Eclipse 以前的版本中这是十分令人讨厌的）。

参考

5.3 节，建立调试会话；5.4 节，设置断点；5.5 节，单步调试代码；5.7 节，运行选定的代码行。

5.7 运行选定的代码行问题

你想在没有设置断点的特定代码行停止执行。

解决方案

选择 Run → Run to Line。

讨论

除了遇到断点后停止执行之外，还可以使用 Run → Run to Line 菜单项。只需单击一行，并选择 Run → Run to Line，代码将继续执行，直至到达该行。当调试要跳过大段的代码并快速设置特别的断点时，这一菜单项是非常有用的。

参考

5.3 节，建立调试会话；5.4 节，设置断点；5.5 节，单步调试代码；5.6 节，在遇到断点前继续运行。

5.8 监视表达式和变量问题

你需要在代码执行时监视变量或表达式的值。

解决方案

在调试时选中变量或表达式，右击它，并选择 Watch，或选择 Run → Watch。从这时起，该变量或表达式以及它的值，将出现在 Expressions 视图中。

讨论

在图 5-15 中可以看到一个例子，在这里，我们把变量 `loopIndex` 的值添加到了 Expressions 视图中。在调试器中执行代码时，变量 `loopIndex` 的当前值总是出现在 Expressions 视图中，除非你在 Expressions 视图中右击它，并选择 Remove。

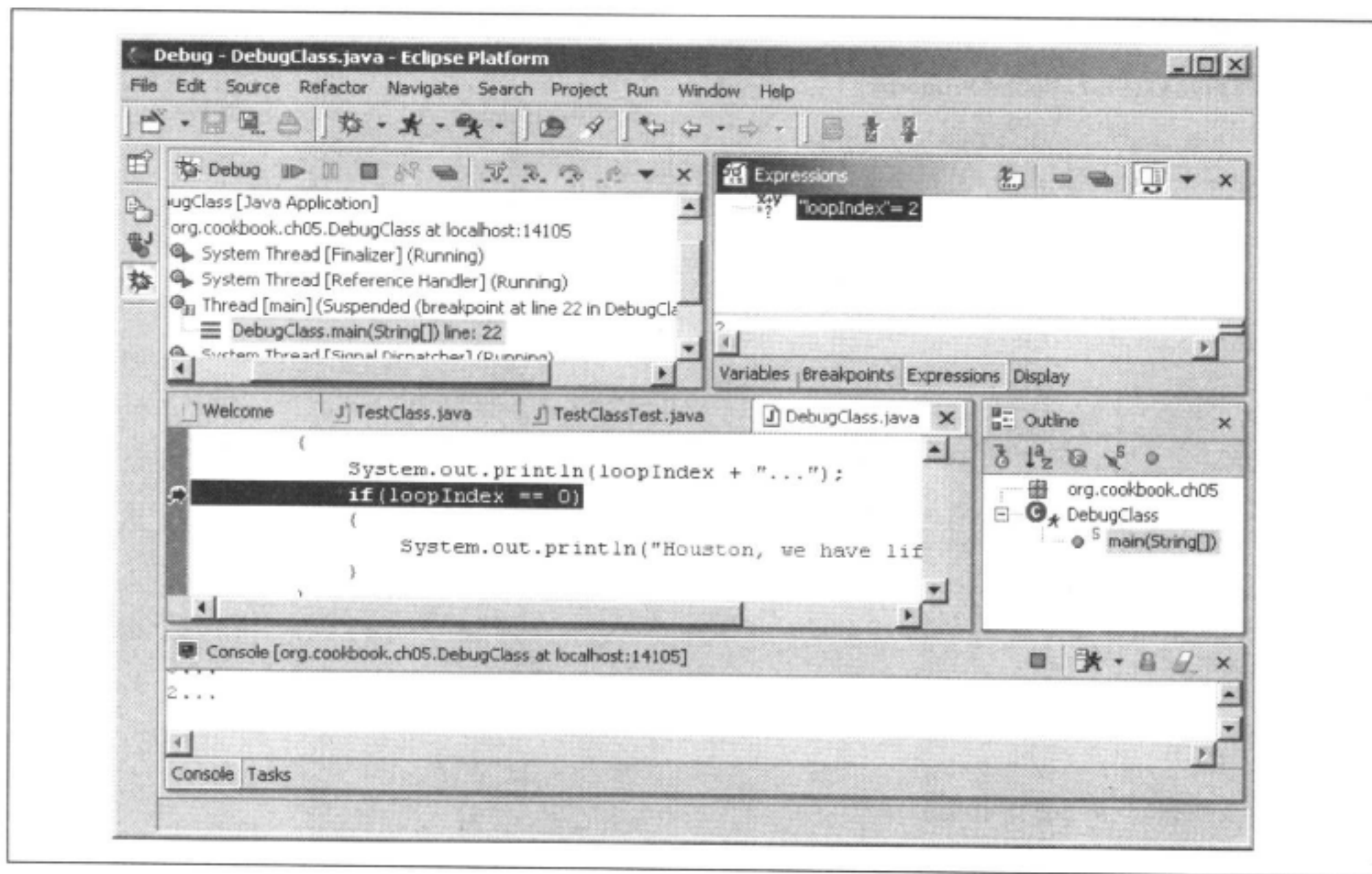


图 5-15：监视变量的值

5.9 为断点设置命中计数 问题

你想仅当遇到一个断点指定次数后才中断执行。

解决方案

可以在经过指定的命中次数（称为命中计数）后才启用断点。要设置断点的命中计数，可在 Breakpoints 视图中右击该断点，并单击 Properties，打开 Java Line Breakpoint Properties 对话框。选中 Enable Hit Count 复选框，并输入所需的命中计数。

讨论

在图 5-16 中可以看到一个例子，其中一个断点的命中计数被设置为 3。

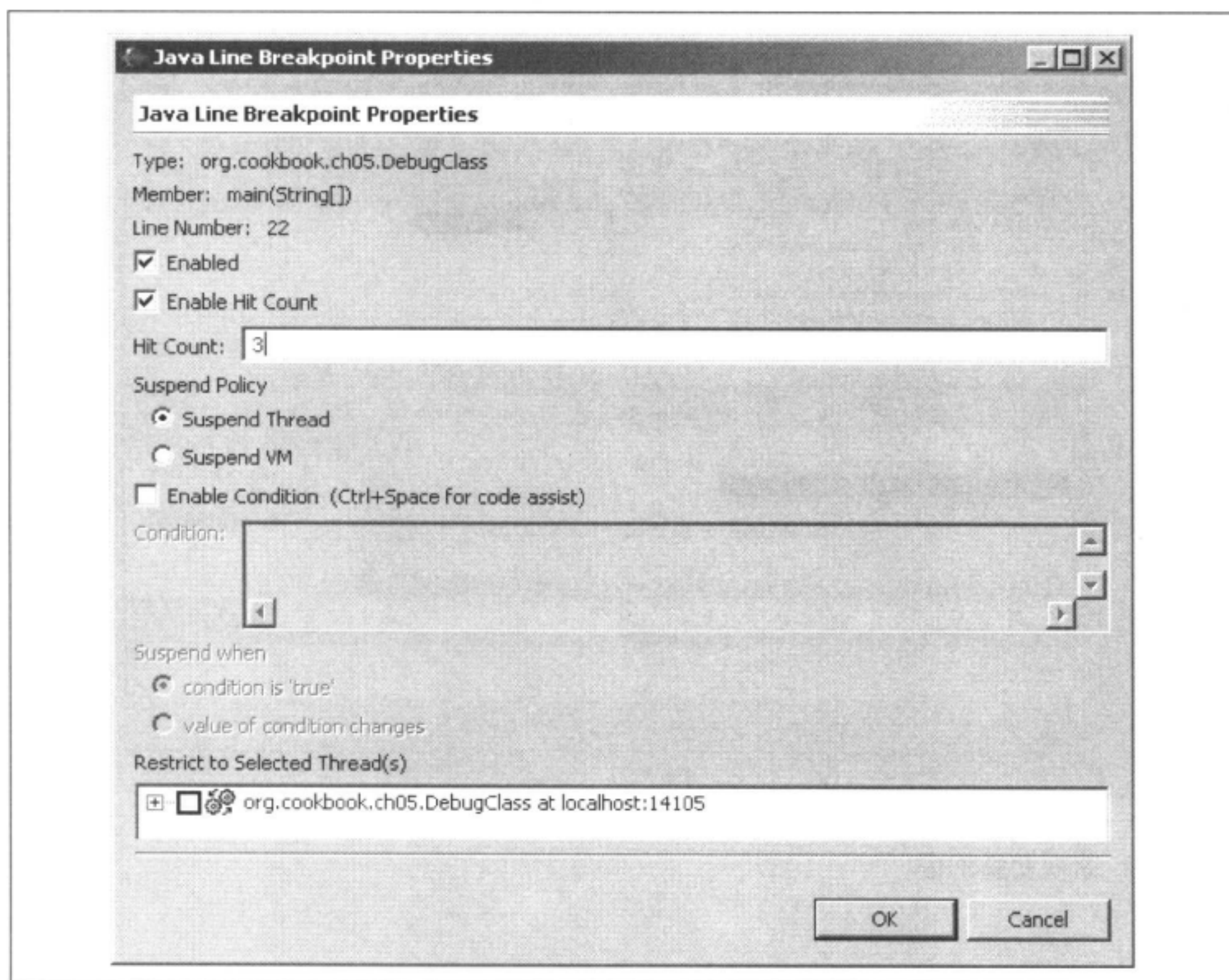


图 5-16：设置断点的命中计数

单击 OK，关闭此对话框，然后重新启动调试会话（可以通过单击 Terminate 按钮或选择 Run → Terminate，来终止调试会话，并再次启动调试会话）。设置完成后，代码的执行将在第三次遇到断点时暂停。

注意：还可以通过在 Breakpoints 视图中右击断点，单击 Hit Count，并在打开的对话框中输入所需的命中计数，来设置断点的命中计数。

参考

5.10 节，配置断点条件。

5.10 配置断点条件

问题

你想在代码中出现某种条件时，比如，当变量等于某个值时，暂停程序的执行。

解决方案

配置断点，使之响应指定的条件。这可以通过以下操作来完成：在 Breakpoints 视图中右击断点，单击 Properties，选中 Enable Condition 复选框，并输入要采用的条件（如 `loopIndex == 2`）。

讨论

为了说明其工作原理，在图 5-17 中，我们把一个断点的条件设置为 `loopIndex == 2`。

现在，当遇到断点，且变量 `loopIndex` 的值等于 2 时，断点将被激活。

还可以在某种条件的值改变时中断执行。例如，如果代码的某一部分改变了变量 `loopIndex` 的值，且这种改变是不应当出现的，则可以在 Condition 文本框中输入变量名，并选中 `value of condition changes` 复选框（可以在 Condition 文本框中输入任何有效的表达式，而不仅仅是变量名）。

参考

5.9 节，为断点设置命中计数；5.8 节，监视表达式和变量。

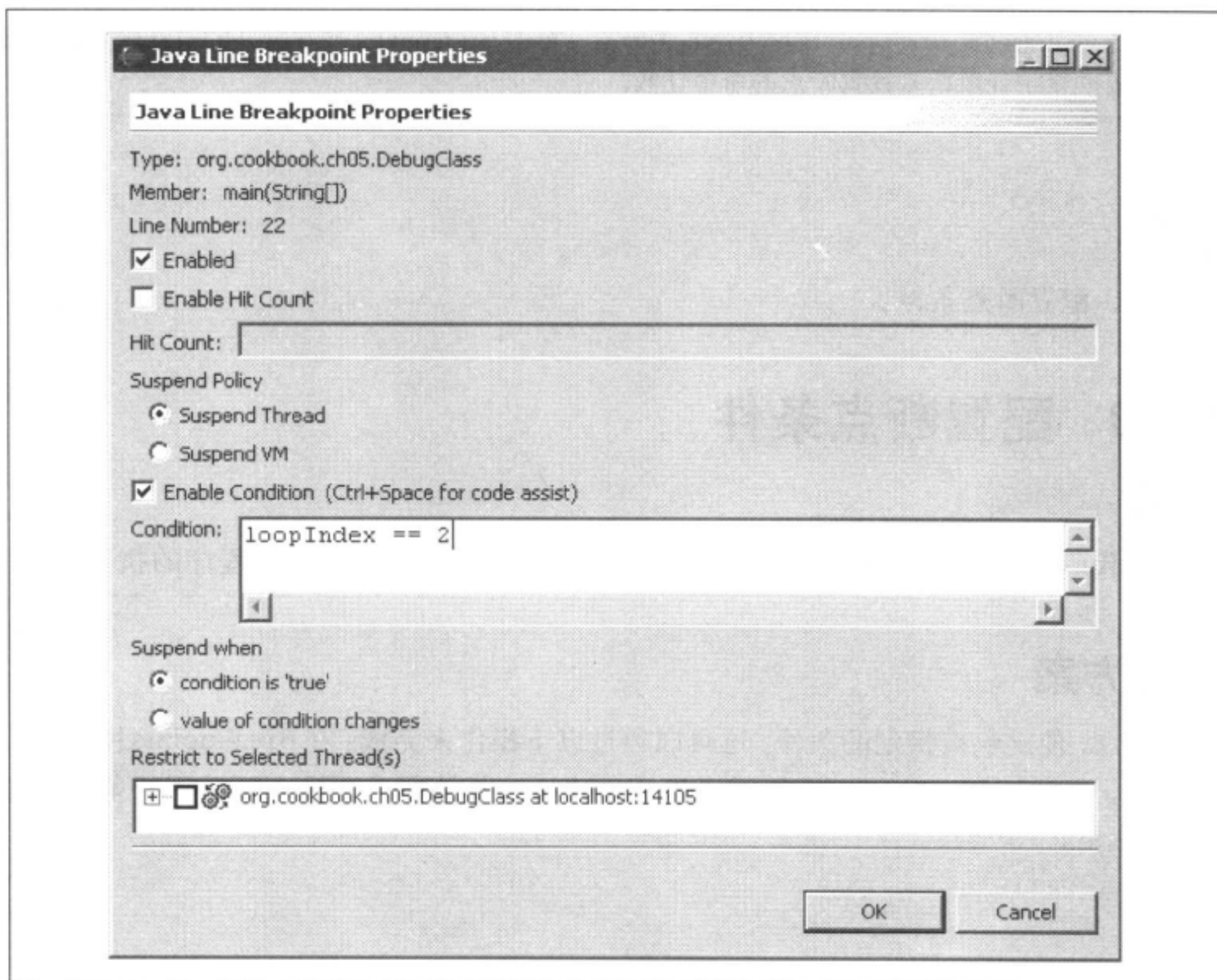


图 5-17：设置断点的条件

5.11 创建字段、方法和异常断点

问题

你需要在即将访问某个字段或方法时，或出现某种异常时，停止执行。

解决方案

使用 Run 菜单中的相应菜单项设置字段、方法或异常断点。

讨论

我们一直在使用的标准断点叫做行断点；除了行断点以外，JDT 编辑器还支持 3 种其他类型的断点：字段断点、方法断点和异常断点。

字段断点

字段断点，也叫做监视点，在代码即将访问和/或修改一个字段的值时，暂停执行（不能对局部变量设置监视点，只能对字段设置监视点）。这与在标准断点上设置条件不同；每当要以任何方式访问字段时，就会出现断点。使用监视点通常要比试图捕获代码中所有可能的点容易得多，在后一种情况下要监视的字段可能会被修改。

要设置监视点，可在Java视图中选中一个字段，并选择Run → Add/Remove Watchpoint。新的监视点将出现在Breakpoints视图中，而且可以通过右击监视点，然后单击Breakpoint Properties，打开如图5-18所示的Java Watchpoint Properties对话框，来配置监视点。需要特别注意的是，在这里可以选中两个复选框——Access和Modification——通过这两个复选框，可以指明是否需要在访问和/或修改字段时暂停执行。

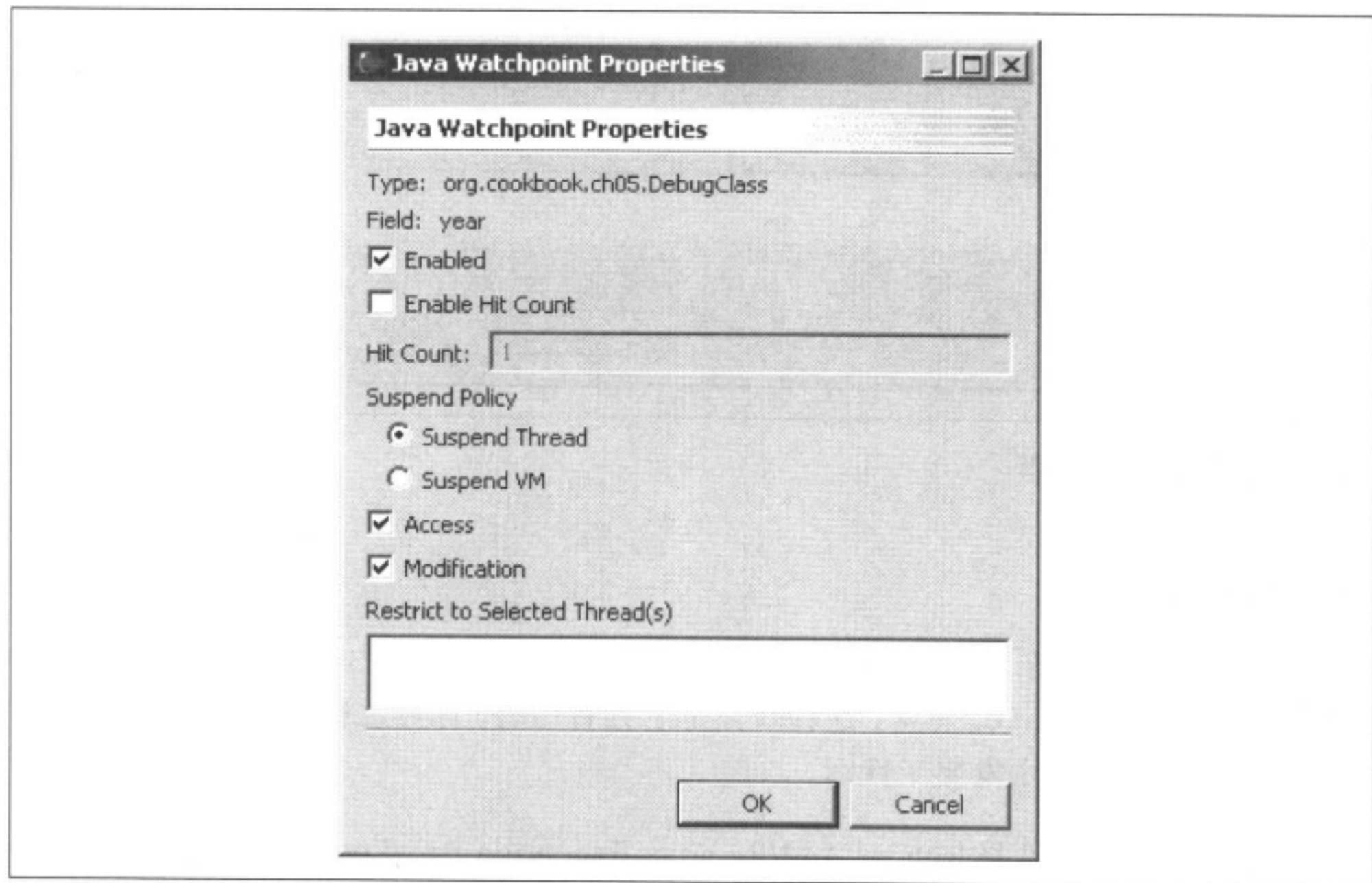


图 5-18：设置监视点属性

方法断点

方法断点在进入或退出一个方法（取决于如何配置断点）时暂停执行。这些断点通常用于你没有源代码的方法上。要设置方法断点，可在Java视图中选中该方法的调用，并选择Run → Add/Remove Method Breakpoint。在Breakpoints视图中右击方法断点，打开如图5-19所示的Java Method Breakpoint Properties对话框，即可配置方法断点。使

用该对话框中的 Entry 和 Exit 复选框，可以选择是在进入方法时还是在退出方法时出现断点，或者在两种情况下都出现断点。

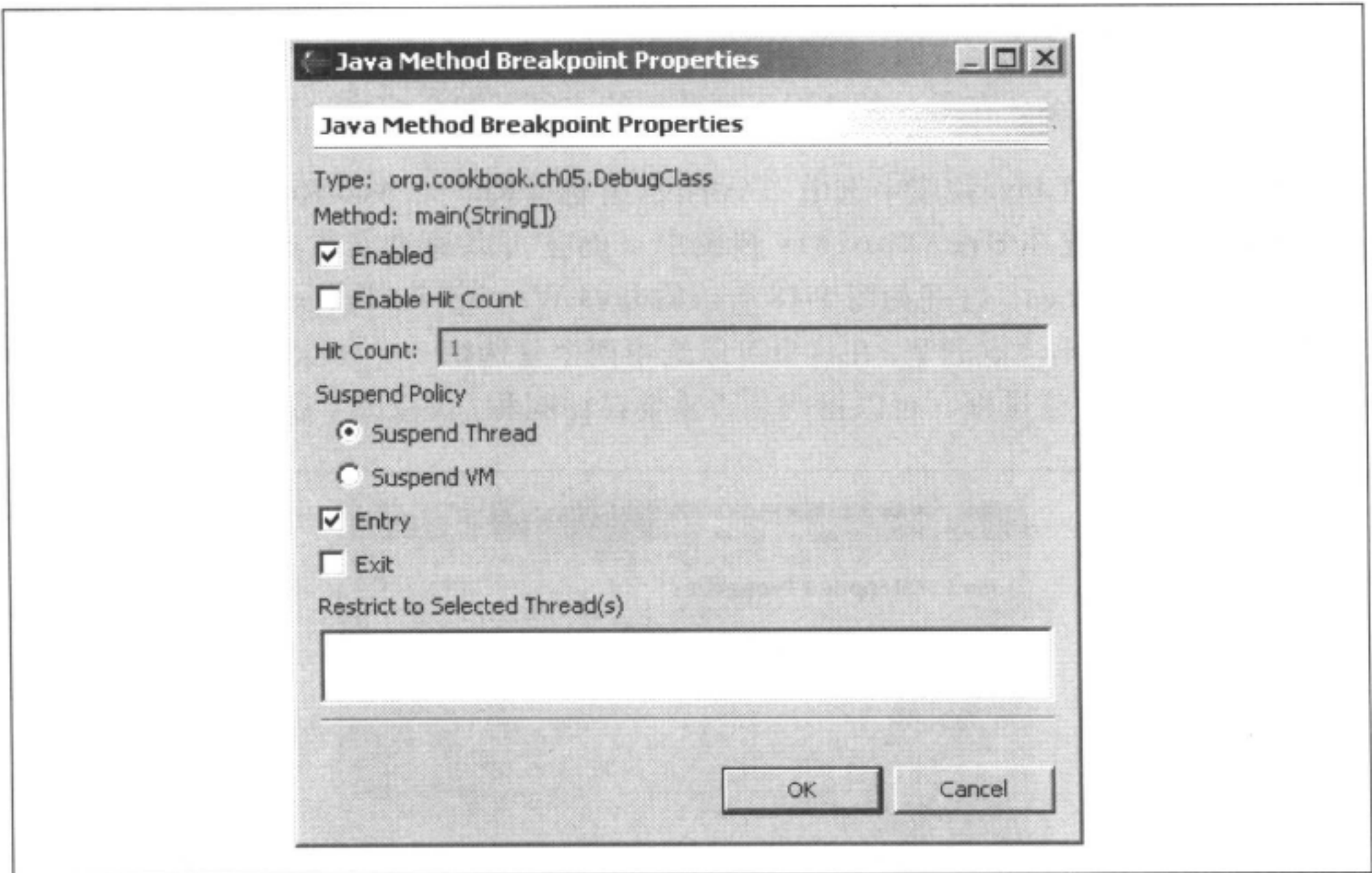


图 5-19：设置方法断点

异常断点

还可以使用异常断点，在发生异常时暂停执行。如果代码引发异常，如 `NullPointerException` 异常，这种断点是十分有用的。你可以暂停执行，查看在捕获或未捕获异常时代码中发生了什么。

要设置异常断点，可选择 `Run → Add/Remove Exception Breakpoint`，打开如图 5-20 所示的对话框，选择你感兴趣的异常，并选择是在捕获异常时还是在未能捕获异常时暂停执行，或者在两种情况下均暂停执行。

可以像配置其他任何断点一样，配置异常断点的属性。只需在 Breakpoints 视图中右击断点，并单击 `Properties` 即可。例如，通过图 5-21 可以明白如何为未捕获的 `java.lang.NullPointerException` 异常配置异常断点。还可以将该断点限于特定的位置，如图 5-21 所示，甚至设置命中计数。

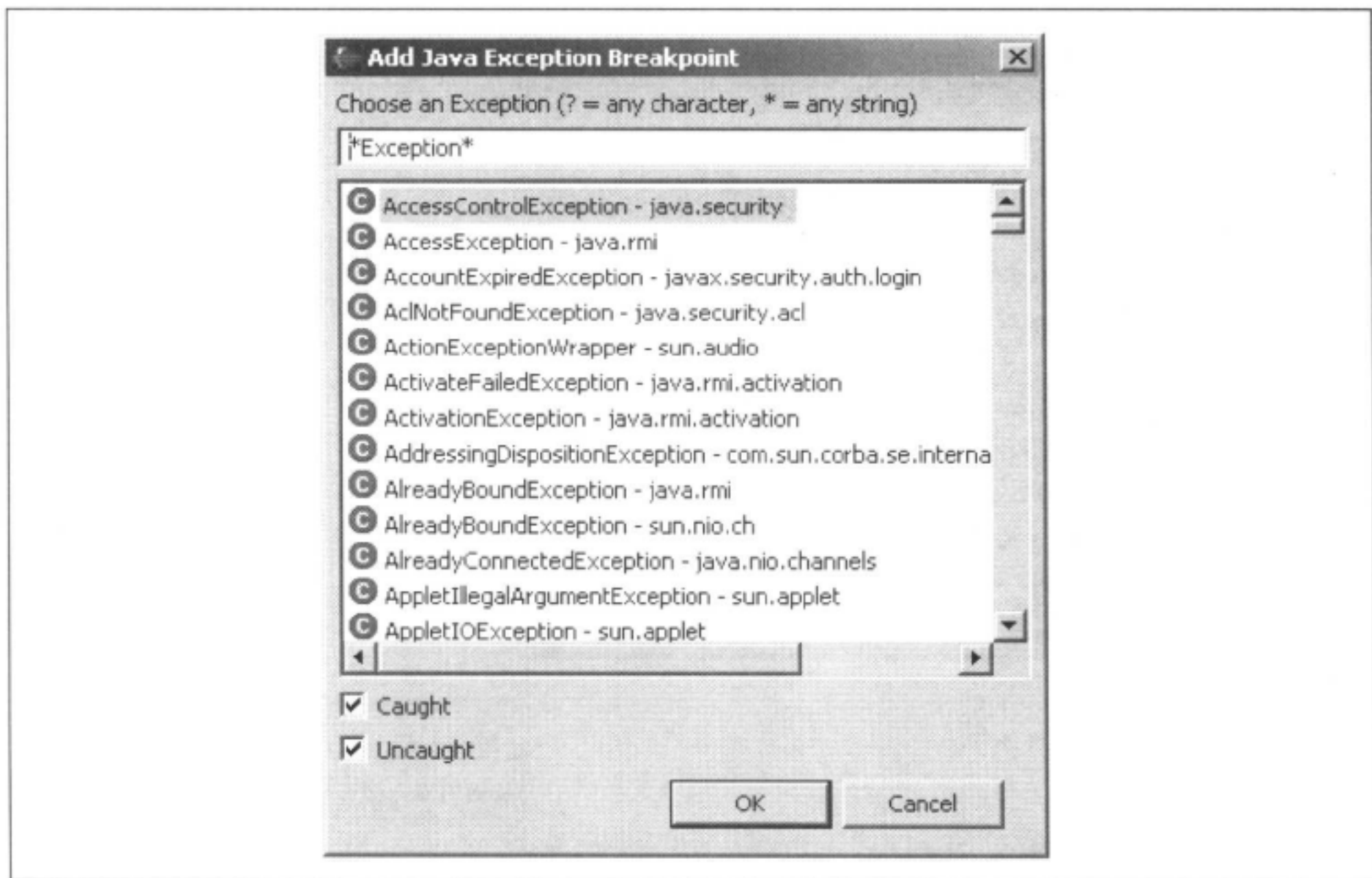


图 5-20：设置异常断点

Eclipse 3.0

在Eclipse 3.0创建断点的过程与在以前的版本中相同。但在Eclipse 3.0中，Add/Remove Watchpoint 菜单项改称为 Toggle Watchpoint。另外，断点属性对话框的布局也有所不同（但仍然包含相同的项目）。

参考

5.10 节，配置断点条件；*Eclipse*（O'Reilly）一书的第 3 章。

5.12 计算表达式的值

问题

你想在调试代码时计算一个表达式的值。

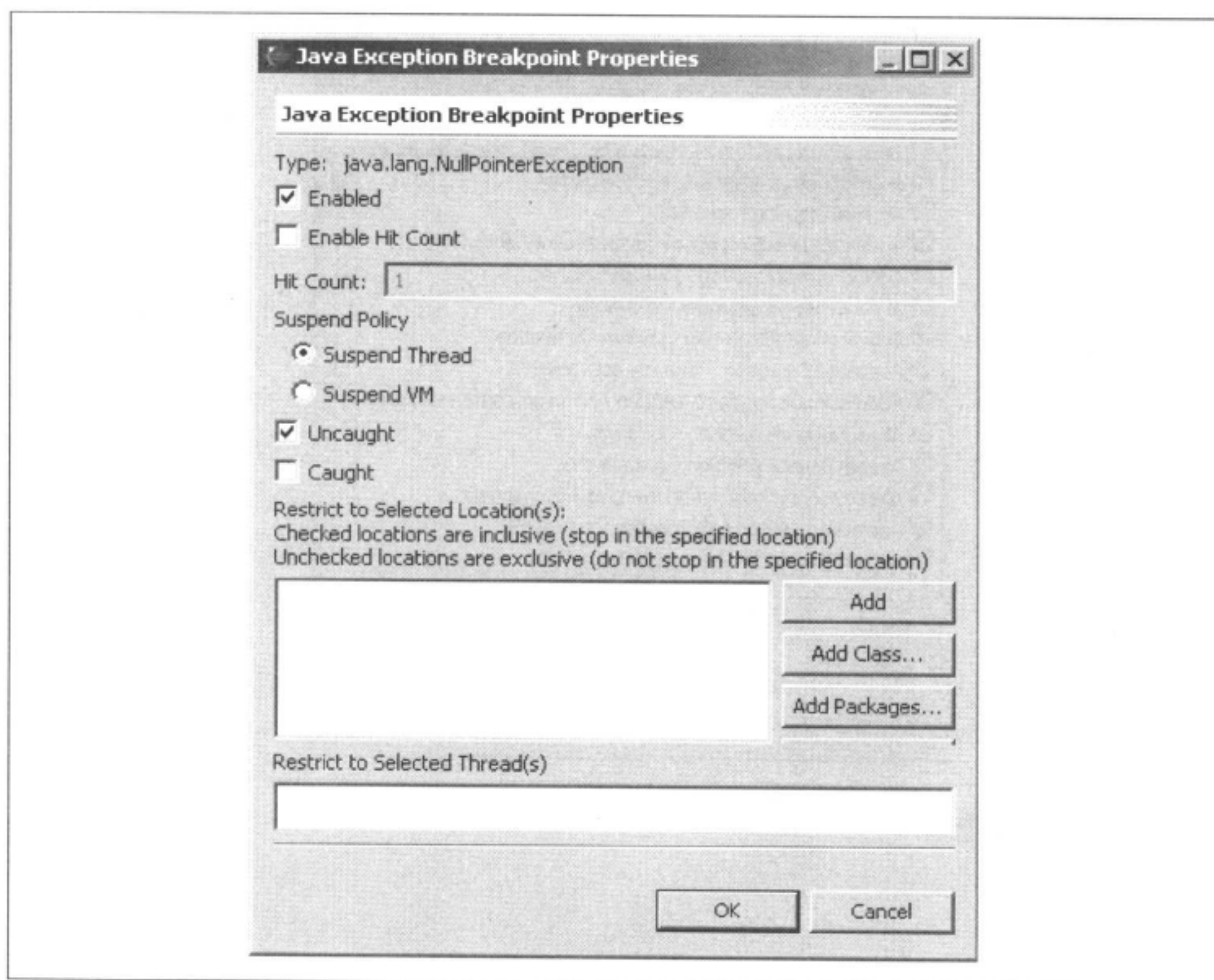


图 5-21：设置 Java 异常断点的属性

解决方案

在 Debug 透视图的 Expressions 视图中，输入要计算的表达式，右击表达式，并单击 Inspect。

讨论

在调试会话期间，计算表达式的值可能是十分有用的。例如，假设代码中有一个名为 temperature 的变量，设置为华氏 72 度：

```
public class DebugClass
{
    public static int temperature = 72;
    .
    .
    .
}
```

在调试代码以检查其与你从欧洲获得的新代码的接口时,突然意识到需要使用摄氏温度。如何进行快速转换呢?为了在调试期间在 Expressions 视图中打开 temperature 变量,可在代码中选中它,并单击 Inspect。在 Expressions 视图中可以看到,temperature 的当前值等于 72,如图 5-22 所示。现在,在 Expressions 视图底部的详细窗格中,输入要把温度值转换为摄氏温度的表达式,即 $(\text{temperature} - 32) * 5 / 9$ 。然后选中该表达式,右击它,并单击 Inspect。这个表达式被添加到 Expressions 视图的表达式列表中,且其当前值等于 22,如图 5-22 所示。另外,表达式的值将随着代码的执行自动更新。

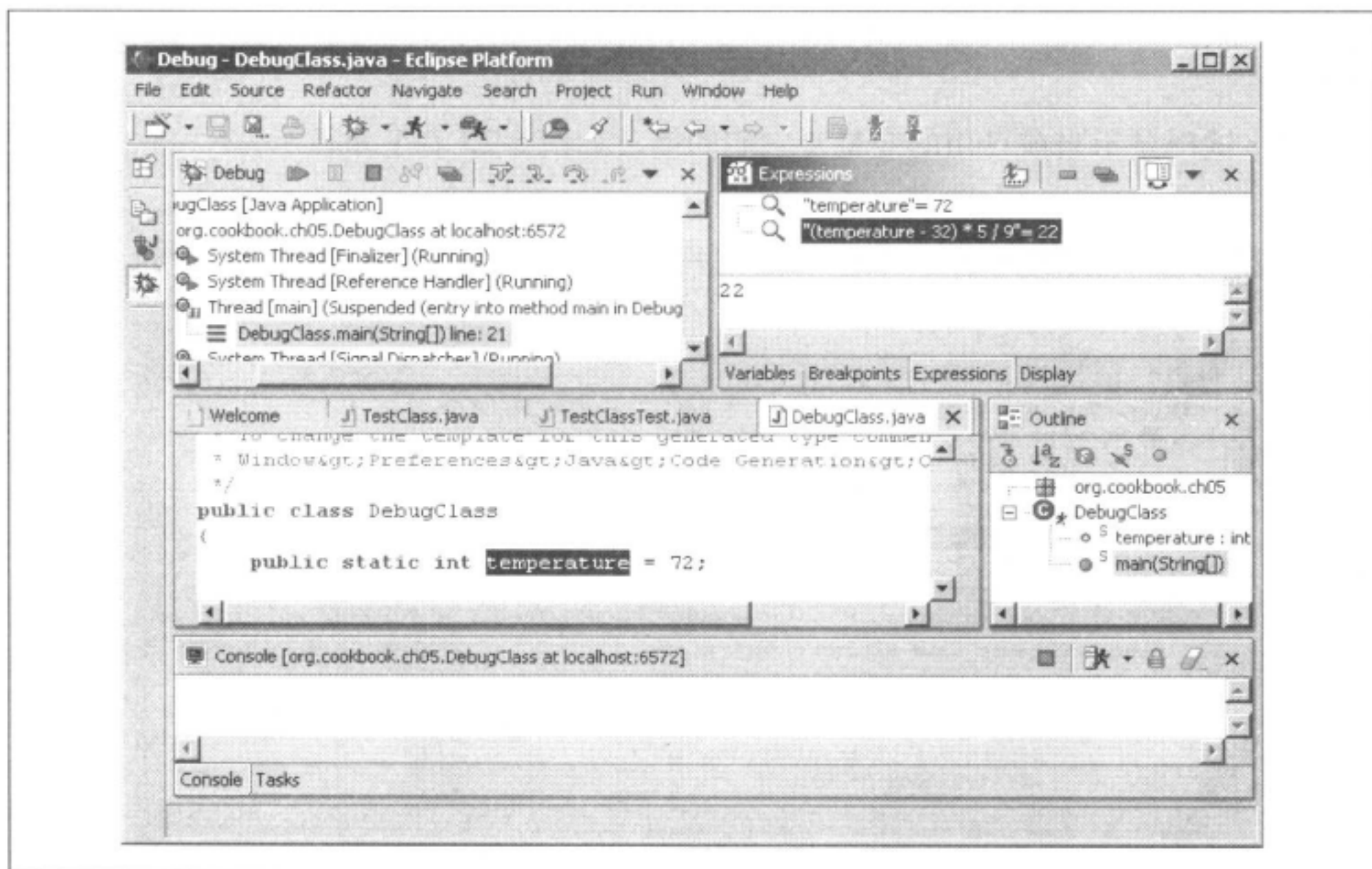


图 5-22: 计算表达式的值

Eclipse 3.0

在 Eclipse 3.0 中,默认情况下,详细窗格将出现在 Expressions 视图的右侧,而不是底部。

参考

5.3 节, 启动调试会话; 5.13 节, 在调试期间为变量赋值。

5.13 在调试期间为变量赋值

问题

在调试代码时，你想改变变量的值，以便对新值进行测试。

解决方案

只需在 Variables 视图中双击字段名或变量名，然后在打开的对话框中输入新值即可。

讨论

Variables 视图显示当前可用的变量及它们的值，如图 5-23 右上角所示。

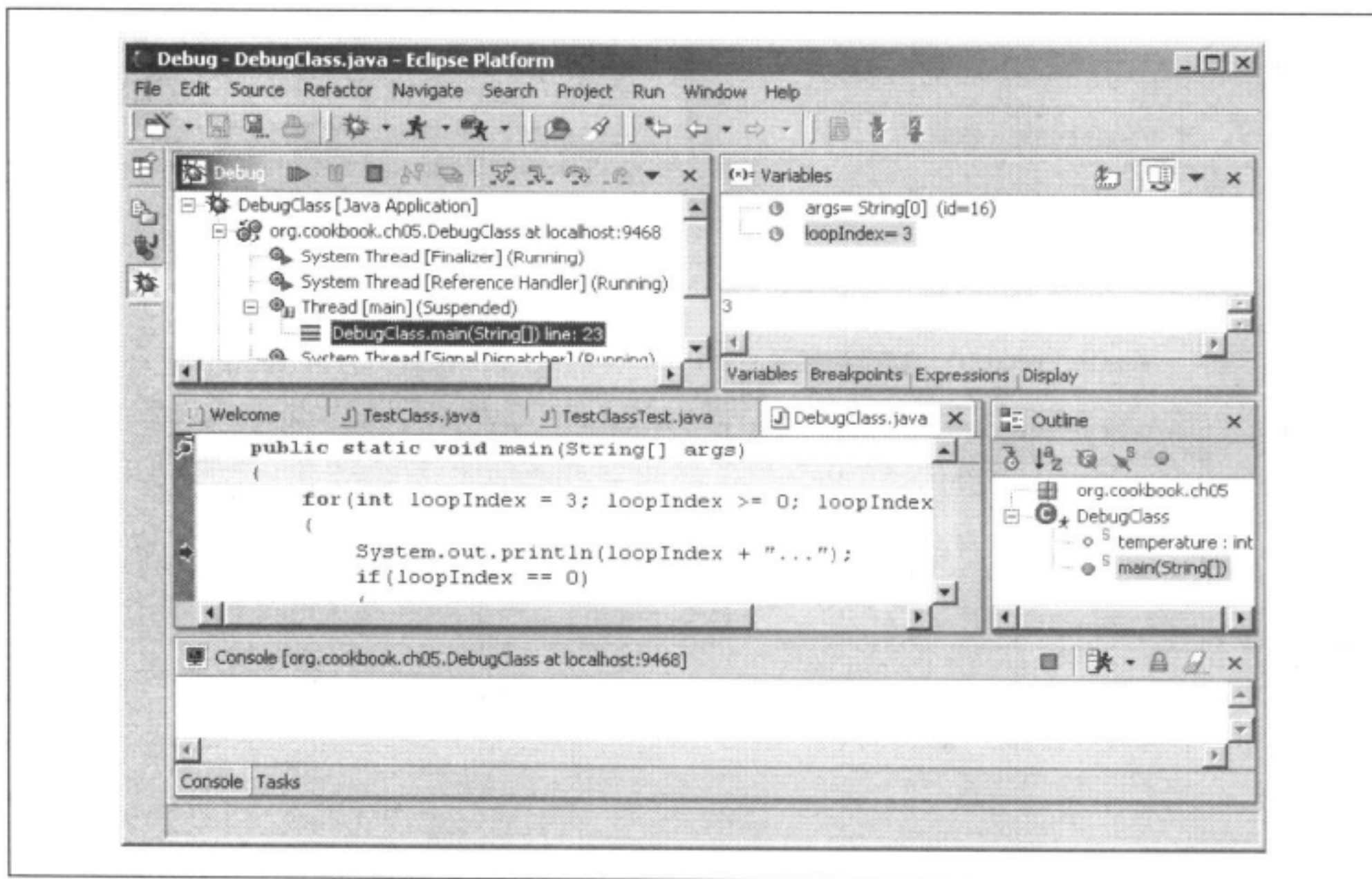


图 5-23：使用 Variables 视图进行调试

要修改变量的值，只需单击变量，并输入一个新值即可。例如，要把变量 `loopIndex` 的值从 3 改变为 2，可在 Variables 视图中双击 `loopIndex`，在打开的对话框中输入 2（如图 5-24 所示），并单击 OK 即可。

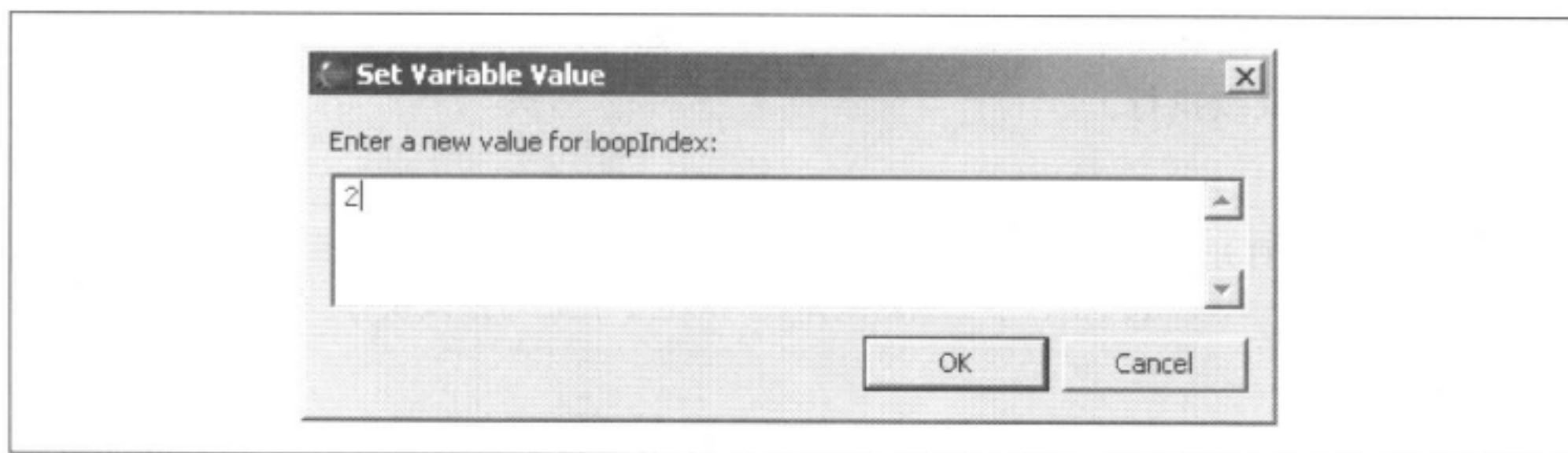


图 5-24：快速修改变量的值

然而，需要注意的是，在运行的代码中乱改数据可能会中断代码的执行。

Eclipse 3.0

在 Eclipse 3.0 中，使用变量过滤器，可以以更紧凑、更有意义的形式显示某些 Java 类型，如集合和映像。用 Variables 视图中的切换按钮可以控制这些过滤器。

参考

5.12 节，计算表达式的值。

5.14 快速修改代码

问题

你发现了代码中的一个错误，你想修复它并测试修复，且不必重启调试会话。

解决方案

如果你有 Java 1.4 或更新的版本，只需在调试时编辑代码并保存代码即可（前提是打开 auto-build on save 功能），而且可以继续调试。

讨论

调试的同时编辑代码的过程叫做热码替换，而且需要使用 Java 1.4 或更新的版本。务必要打开 auto-build on save 功能（默认情况下该功能是打开的）。如果没有打开，可通过选择 Window → Preferences → Workbench，并选中 Perform build automatically on resource modification 复选框，来打开此功能。

现在，调试代码，直至执行在一个断点处停下来。此时，可以自由地编辑代码。编辑之后，保存代码并继续执行。

注意：如果你不想打开 auto-build on save 功能，只需在编辑之后选择 Project → Rebuild Project 或 Project → Rebuild All，手工重新编译代码即可，然后可继续执行。

注意，必须使用支持热码替换的 Java 1.4 或更新版本。如果实际开发的代码是面向较早的 Java 版本，在 Eclipse 中可以使用不同的 JRE 来运行和调试代码。如果需要设置调试运行配置，以便使用不同的 JRE，可以选择 Run → Debug，然后单击 JRE 标签，并选择要用于调试的 JRE。

参考

4.8 节，选择 Java 编译运行环境。

使用 Eclipse 进行团队开发

6.0 简介

专业开发人员通常以团队的形式进行工作，而 Eclipse 可以胜任这项任务。针对团队开发，Eclipse 支持并行版本系统（Concurrent Versions System, CVS）。如果你正在进行团队开发，那么必须将你的开发工作与其他人的工作协调，以避免冲突。整个团队共用相同的代码，这意味着你的天才工作可能因某人的粗心而意外毁掉。

源控制可以避免这种问题的发生，因为它以明确定义的方式控制对共享代码的访问。除了控制对代码的访问之外，源控制还维护着一个修改历史记录，以便从较早的版本恢复代码。由于源控制维护着代码的历史记录，所以不仅能够从较早的版本恢复代码，而且可以将当前代码与较早的版本进行比较，以快速了解它们之间的差异。

像 Java 世界中的许多其他项目一样，CVS 也是一个开源项目。CVS 首次出现于 1986 年，当时它是一组 Unix shell 脚本，直到 1989 年专用 CVS 软件首次出现为止。如今，CVS 可以在许多操作系统上使用，从 Unix 和 Linux 到 Windows。

注意：关于 CVS 的详细内容，请访问 <http://www.cvshome.org>。

CVS 储存库是开发人员存储要共享的代码文件的地方。要从储存库中取回一个文件，应从储存库中检出该文件。如果想存储文件的最新修订版，可以将它提交给储存库。刷新储存库中的代码副本的过程叫做更新。

CVS 的术语与 Eclipse 稍有不同；Eclipse 中的项目在 CVS 中称为模块。每一个模块在储

存库中都有自己的目录，便于区分模块。标准项目也叫做物理模块，而逻辑模块或虚拟模块是相关资源的集合。

如何提供代码的许多副本，以便同时检出？这取决于所采用储存库模型。下面是储存库的选项：

Pessimistic locking（悲观锁定）

顺序访问。使用这种锁定时，每次只能有一个开发人员可以检出某个文件。当检出一个文件时，该文件被锁定。其他人可以检出该文件的只读副本，但不能修改原始文件。访问是有序的。

Optimistic locking（乐观锁定）

随机访问。使用这种锁定时，开发人员可以自由地检出和修改文件。当提交修改后的文件时，储存库软件将自动合并所有修改。如果合并操作出现问题，储存库软件将标记这些修改，并请求你解决问题。

默认情况下，CVS使用乐观锁定（有些CVS软件还支持悲观锁定）。本书将使用乐观锁定，这是Eclipse支持的锁定类型。可以使用CVS服务器来处理实际的文件操作，就像本章中所做的一样。

在提交文件时，CVS还可以为每一个文件自动分配一个版本号。首次提交一个文件时，其版本号为1.1（在有些CVS中为1.0）。下一次，版本号为1.2，依此类推。在本地更新代码时，Eclipse不仅覆写文件的本地版本，而且以智能的方式将修改与本地文件合并。如果存在冲突，它将插入特殊的CVS标记，突出显示冲突行，而且在运行代码之前必须解决这些冲突。通常，更新会顺利进行，但如果因对文件做了大量的改动或没有及时更新，致使存在大量冲突，则可能需要花一些时间来解决这些冲突。

CVS还可以支持在同一模块中同时进行多个开发流，称为“分支”（branch）。一个模块中的主开发流称为“头”（head），而分支是从主开发流中分出的叉。例如，一个分支代表项目的一个测试版，或者添加到代码中首次进行测试的某项新功能。

6.1 获得 CVS 服务器

问题

你想开始使用CVS，并需要安装CVS服务器。

解决方案

CVS服务器可能已经安装了；如果没有CVS服务器，可以下载一个。

讨论

现在，大多数 Linux 和 Unix 安装都带有 CVS 服务器，作为其标准发布的一部分。要检查你是否已经有了一个可用的 CVS 安装，可在命令行输入 `cvs --help`；你将看到一个帮助提示列表。如果其中没有包含 CVS 服务器，可以从站点 <http://www.cvshome.org> 下载。在大型系统中，如果找不到 CVS 安装，可以询问支持人员。

如果你使用的操作系统是 Windows，则可以发现有许多 CVS 服务器可供下载。例如，历史悠久的 CVSNT 可以从站点 <http://www.cvsnt.org> 免费下载。只需运行可执行文件，即可安装。

可用的 CVS 服务器有许多种，而且都附有自己的安装说明。这里不打算再重复这些安装说明，以往每当有新的版本——使用完全不同的安装说明——出现时，都会提供安装说明。通常，只要你下载了所需的 CVS 服务器，安装并不难。只是要查看随软件一起下载的安装说明。而且，要记住，如果安装过于复杂，且有些选项不能使用，总会有其他的 CVS 服务器可用。

参考

Eclipse (O'Reilly) 一书的第 4 章。

6.2 创建 CVS 储存库

问题

你需要创建一个 CVS 储存库，以便存储与其他人共享的代码。

解决方案

在 Linux 和 Unix 中，使用命令 `cvs -d path init`，其中 *path* 代表要用作储存库的目录的位置。在 Windows CVSNT 中，使用 Repository 标签页上的 Add 按钮可以添加新的储存库。

讨论

在安装了 CVS 服务器之后，需要重建一个储存库，将共享代码存储于其中。在 Linux 和 Unix 中，可以在命令提示符处输入命令 `cvs -d path init`，其中是储存库的位置。

注意： 在创建储存库时，要记住，*path*的权限和所有权必须能够访问开发团队的所有成员。

在Windows中，CVSNT是作为Windows的一种服务运行的。从“开始”菜单启动CVSNT，并从添加了CVSNT的程序组中选择Service控制面板项，打开如图6-1所示的CVSNT控制面板。在CVSNT控制面板中单击Repositories标签，单击Add按钮，输入新的储存库目录的路径，然后单击OK。在图6-1中，使用目录*c:/repository*作为CVS储存库。

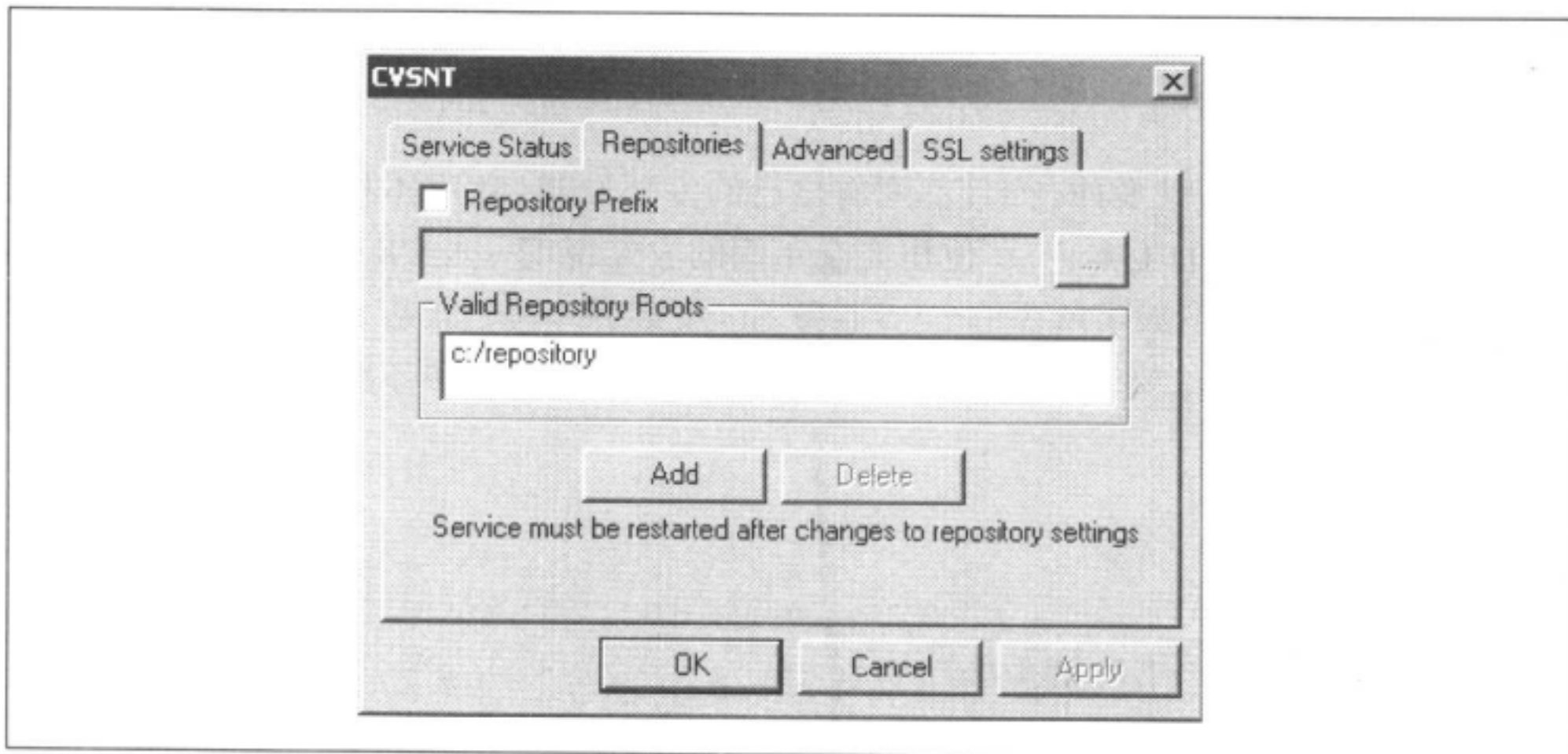


图 6-1：选择储存库

通过从添加了CVSNT的程序组中选择Service控制面板项，打开CVSNT控制面板，可以启动CVSNT服务器。单击CVS Service和CVS Lock Service选项组中的Start按钮，使CVSNT在这两个选项组中显示“Running”字样，如图6-2所示。

参考

6.4 节，将项目存储在CVS储存库中。

6.3 将 Eclipse 连接至 CVS 储存库

问题

你想将Eclipse连接至一个CVS储存库。

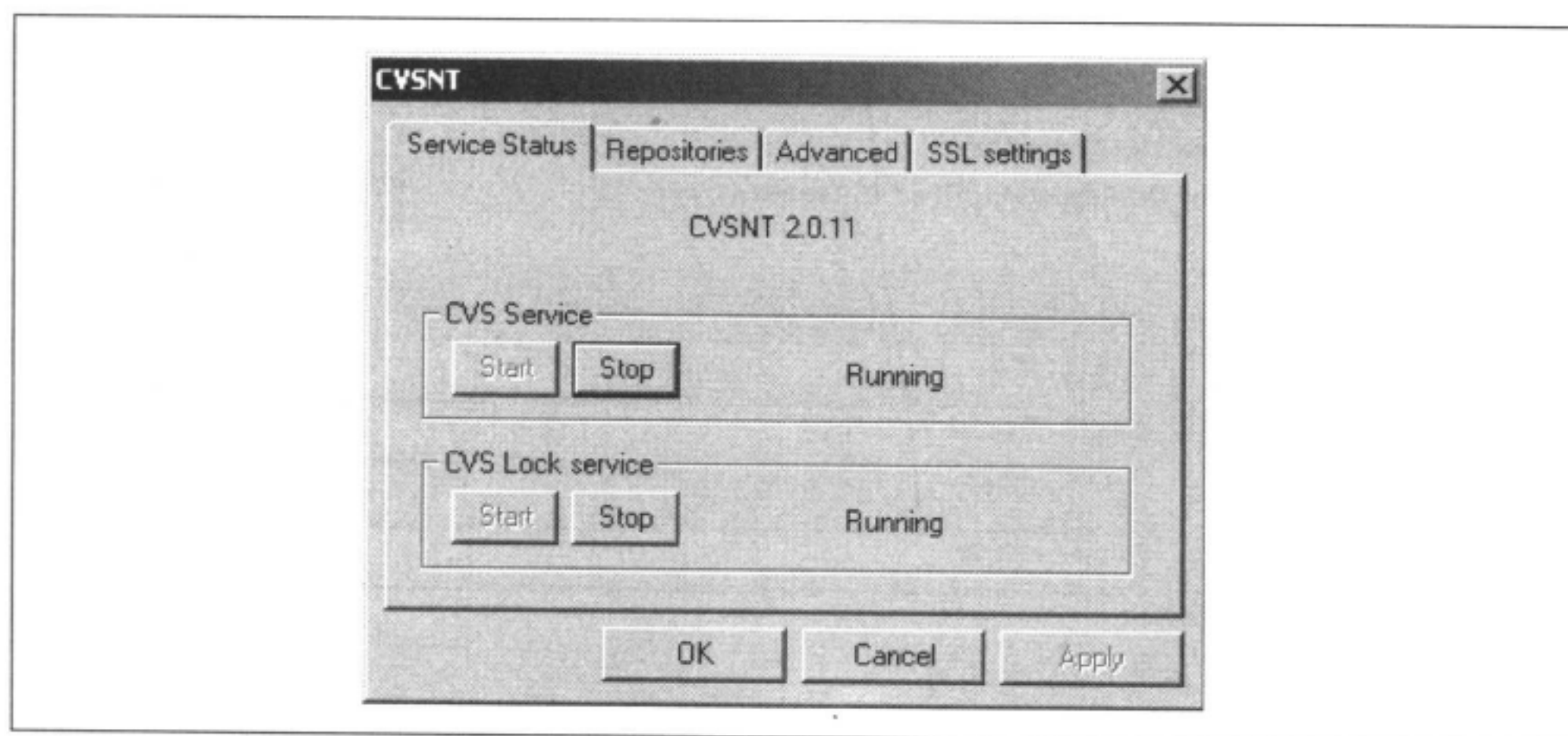


图 6-2：运行 CVSNT

解决方案

在 Eclipse 中，打开 Repositories 视图，右击该视图，并选择 New → Repository Location，打开 Add CVS Repository 对话框。输入所需的信息，然后单击 OK。

讨论

在使用储存库之前，必须通过 CVS 服务器建立 Eclipse 至 CVS 储存库的连接。首先，确保 CVS 服务器处于运行状态。

要将 Eclipse 连接至 CVS 储存库，可选择 Window → Open Perspective → Other，并选择 CVS Repository Exploring 透视图。在首次完成此操作后，Eclipse 把该透视图添加到 Window → Open Perspective 子菜单中，同时将该透视图的快捷方式添加到 Eclipse 窗口的最左边，与其他透视图的快捷方式放在一起。

在 CVS Repository Exploring 透视图打开时，右击该 CVS 透视图左边的空白处，并选择 New → Repository Location，打开 Add CVS Repository 对话框，如图 6-3 所示。

在 Add CVS Repository 对话框中，输入 CVS 服务器的名称，通常是 CVS 服务器的主机名，并输入至 CVS 储存库的路径。要连接至 CVS 服务器，还需要提供用户名和密码，如图 6-3 所示（在本例中，我们使用集成的 Windows NT 安全，所以无需密码）。可以使用两个连接协议与 CVS 服务器进行连接，分别是 SSH (secure shell) 和 pserver。本例中使用 pserver。

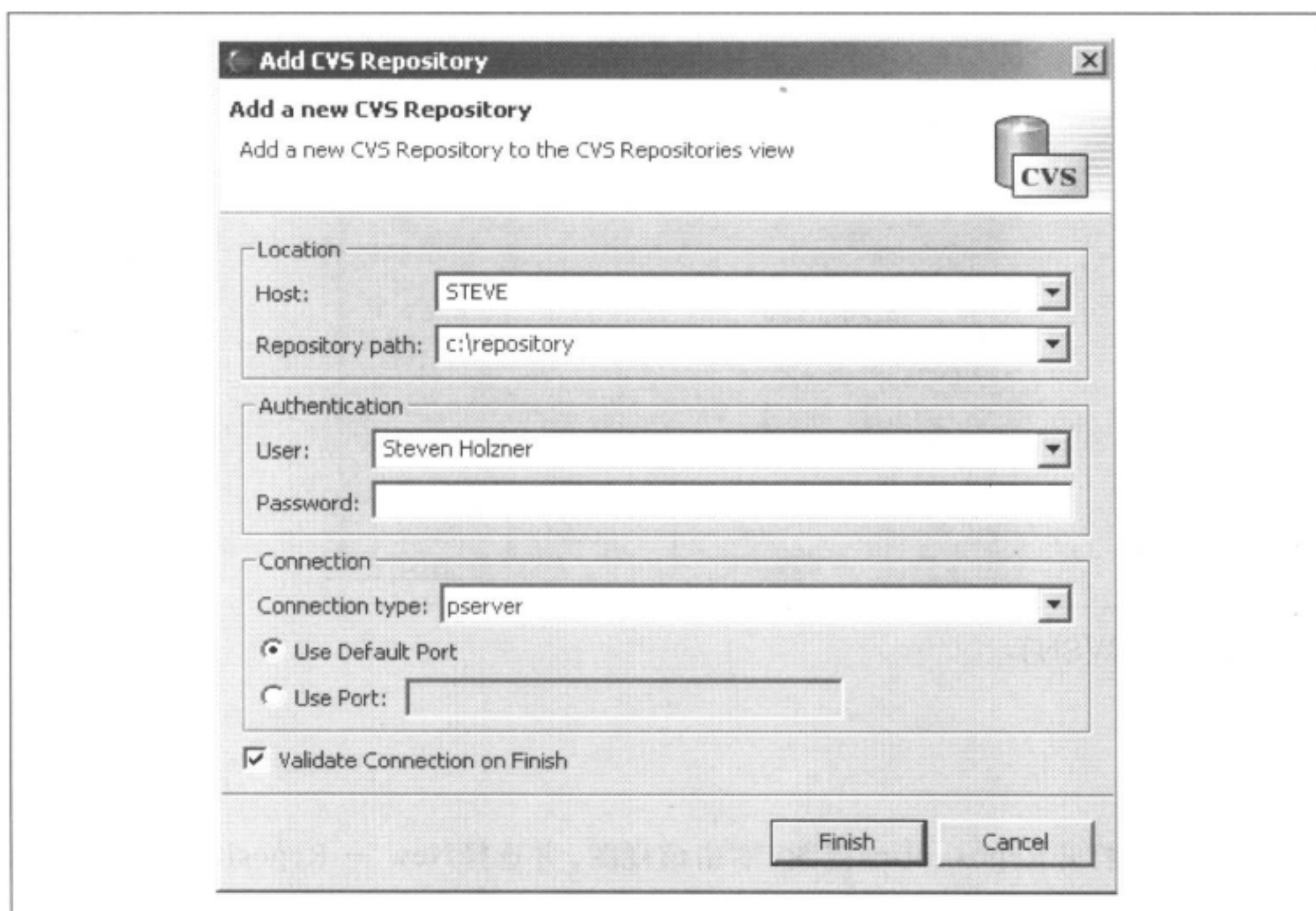


图 6-3：将 Eclipse 连接至 CVS 储存库

注意： `pserver` 是一个 CVS 客户/服务器协议，它使用自己的密码文件和连接。其效率高于 SSH，但安全性不如 SSH。如果安全性是一个问题，应使用 SSH 协议。

在配置了连接之后单击 Finish。新建的到 CVS 服务器的连接应出现在 CVS Repositories 视图中，如图 6-4 所示。

注意： 一个公用的 CVS 服务器可以使用，通过它可以访问 Eclipse 使用的代码；访问 `:pserver:anonymous@dev.eclipse.org:/home/eclipse` 即可。

如果你愿意，可以查看 Eclipse 发送给 CVS 服务器的命令。为此，可选择 Window → Show View → Other → CVS → CVS Console，打开 CVS 控制台。CVS Console 视图将出现（该视图盖住标准的 Console 视图）。

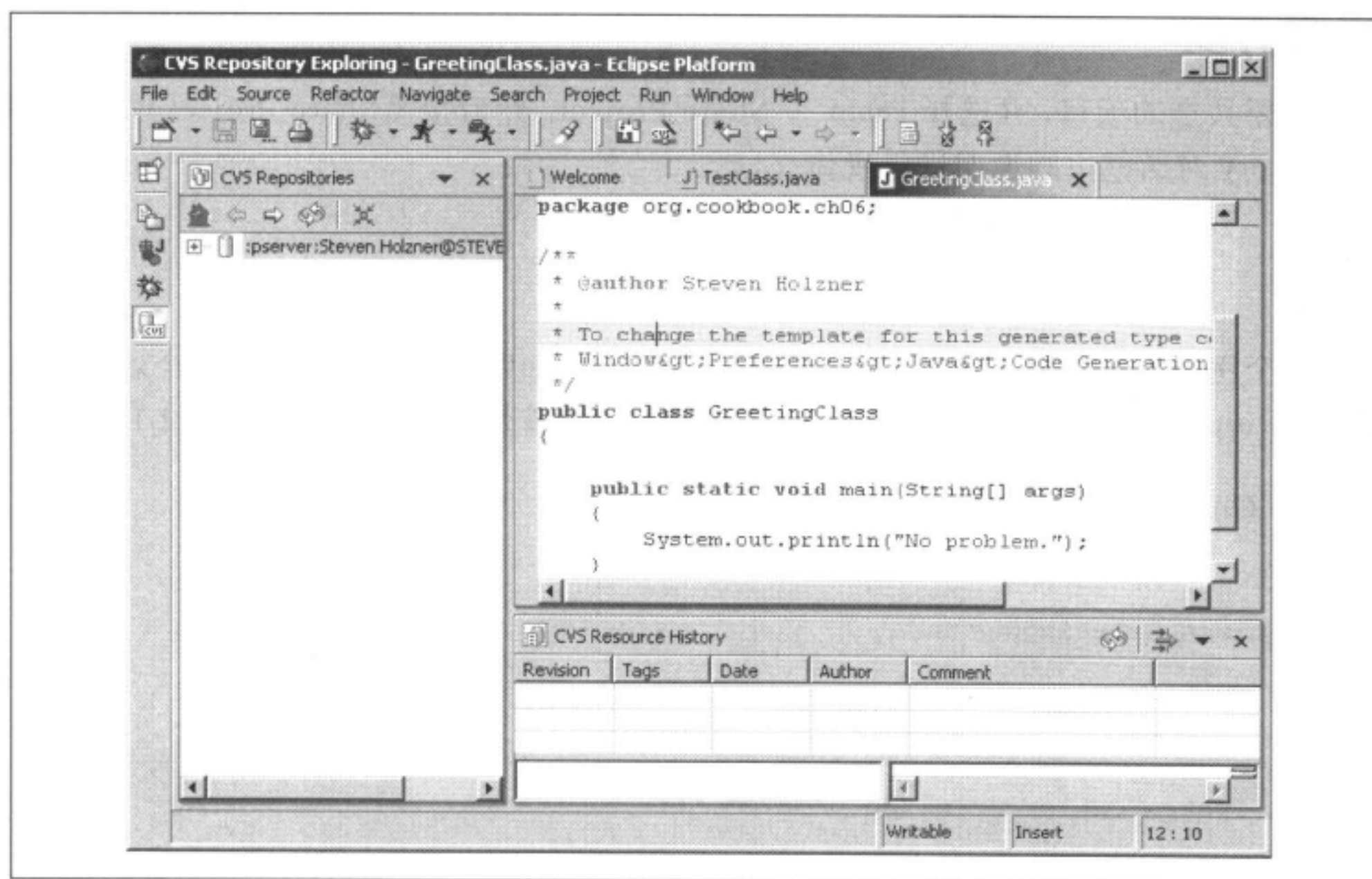


图 6-4: 在 CVS Repositories 视图中新建一个储存库

Eclipse 3.0

除了 pserver 和 SSH 协议以外，Eclipse 3.0 还支持 CVS SSH2 协议。通过 SSH2 Connection Method 参数页可以启用 SSH2 协议（右击一个项目，然后选择 Team → CVS → SSH2 Connection Method）。此后，所有 extssh 类型的 CVS 服务器连接都将使用 SSH2 协议。

参考

Eclipse (O'Reilly) 一书的第 4 章。

6.4 将 Eclipse 项目储存在 CVS 储存库中

问题

你有一个 Eclipse 项目，想存储在一个 CVS 储存库中，供其他开发人员使用。

解决方案

右击需要共享的项目，并选择 Team → Share Project。然后按照 Share Project with CVS Repository 对话框中的说明进行操作。

讨论

作为一个例子，在此我们将创建一个项目，并将其添加到 CVS 储存库中。这个示例项目即 *GreetingApp* 的代码如例 6-1 所示。该代码的任务就是显示消息 “No problem.”。

例 6-1: GreetingApp 项目

```
package org.cookbook.ch06;

public class GreetingClass
{
    public static void main(String[] args)
    {
        System.out.println("No problem.");
    }
}
```

为了将这个项目添加到 CVS 储存库中，打开 Java 透视图，右击该项目，选择 Team → Share Project。这将打开 Share Project with CVS Repository 对话框，如图 6-5 所示。

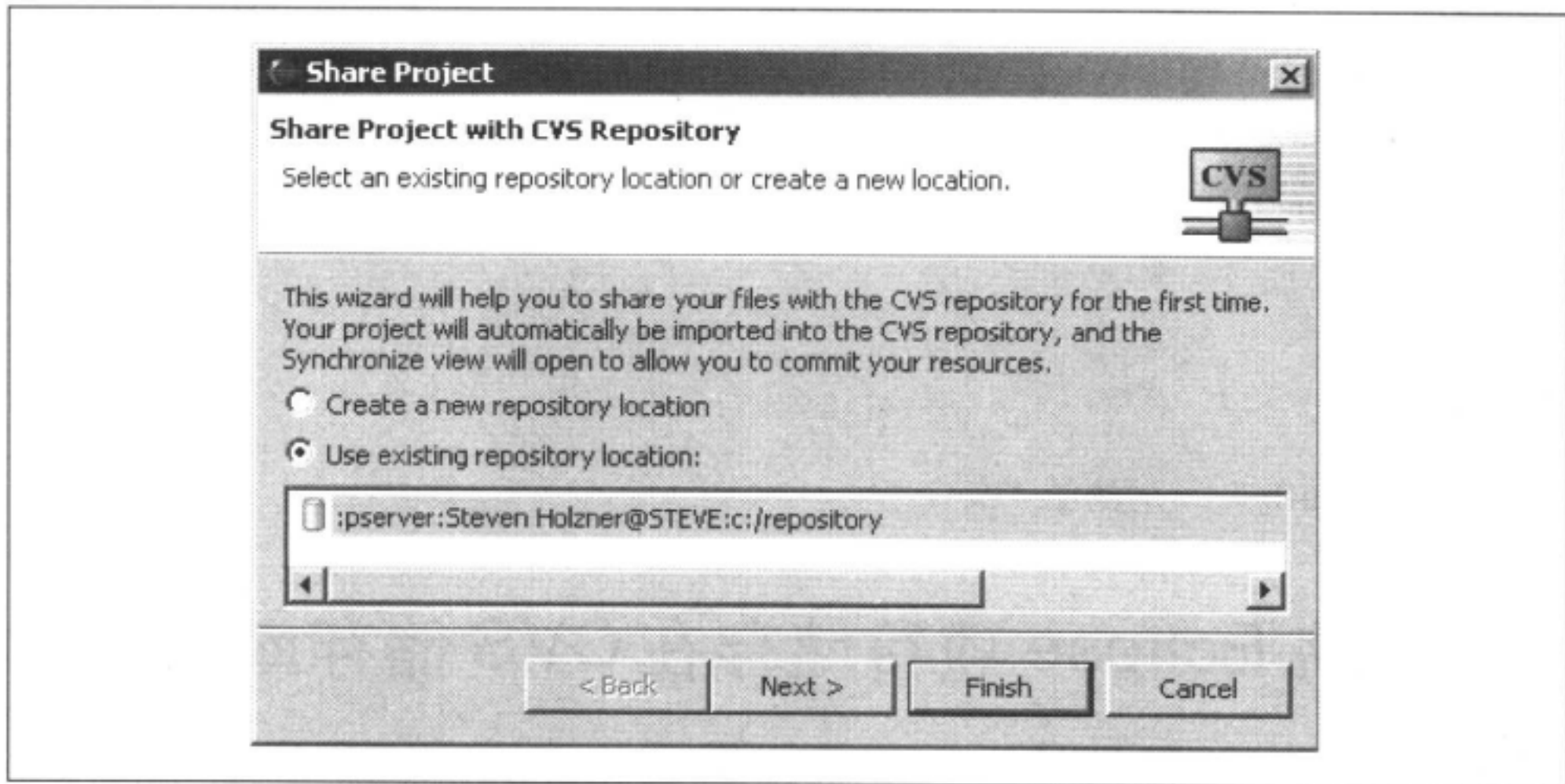


图 6-5: 通过 CVS 储存库共享项目

确保选中 Use existing repository location 单选按钮，并选择要使用的储存库。单击 Finish，将项目添加到储存库中。这将创建一个与 Eclipse 项目同名的 CVS 模块。

注意：如果你想为创建的 CVS 模块起一个不同的名字，单击 Next 而不是 Finish，输入要创建的 CVS 模块的名字，然后单击 Finish。

这会将项目添加到 CVS 储存库中，同时在 Eclipse 中打开 Synchronize 视图，它与 Console 视图重叠在一起（本章稍后将进一步介绍如何使用 Synchronize 视图，因为此时没有什么需要进行同步，现在还用不着该视图）。

参考

6.5 节，将文件提交到 CVS 储存库中。

6.5 将文件提交到 CVS 储存库中

问题

你编辑了一个文件，保存了所做的修改，并且想把它发送至 CVS 储存库中，以便其他人可以访问它。

解决方案

右击文件，并选择 Team → Commit。

讨论

在 6.4 节，我们介绍了如何将项目添加到 CVS 储存库中。为了共享代码，必须检入 (check in) 代码文件。这需要两个步骤：首先，将文件添加到 CVS 储存库中，即将文件向 CVS 服务器注册，但文件并未实际上传；然后，提交文件，即将文件上传到 CVS 储存库中。

从技术上讲，将文件发送到 CVS 储存库中的方法是将文件添加到 Eclipse 的版本控制中，并提交文件。这可以通过以下操作来完成：选择 Team → Add to Version Control，然后选择 Team → Commit，提交文件。

然而，Eclipse 提供了一种简便的方式。要提交一个项目中的所有文件，可右击项目，并选择 Team → Commit。此时，Eclipse 将显示 Add to CVS Version Control 对话框。单击 Details 按钮，并选中与要添加到 CVS 版本控制中的文件相对应的复选框；Eclipse 将列出项目中的所有文件，包括 Java 源文件、.project 文件和 .classpath 文件。然后单击 Yes。

注意： 如果需要作为 Eclipse 项目检入、检出项目，务必要提交 `.project` 文件和 `.classpath` 文件。

Eclipse 将提示你为提交的文件添加注释，以便标记这些文件。在本例中，只需输入一些文本，如 “The Greeting App”，如图 6-6 所示，并单击 OK 即可。

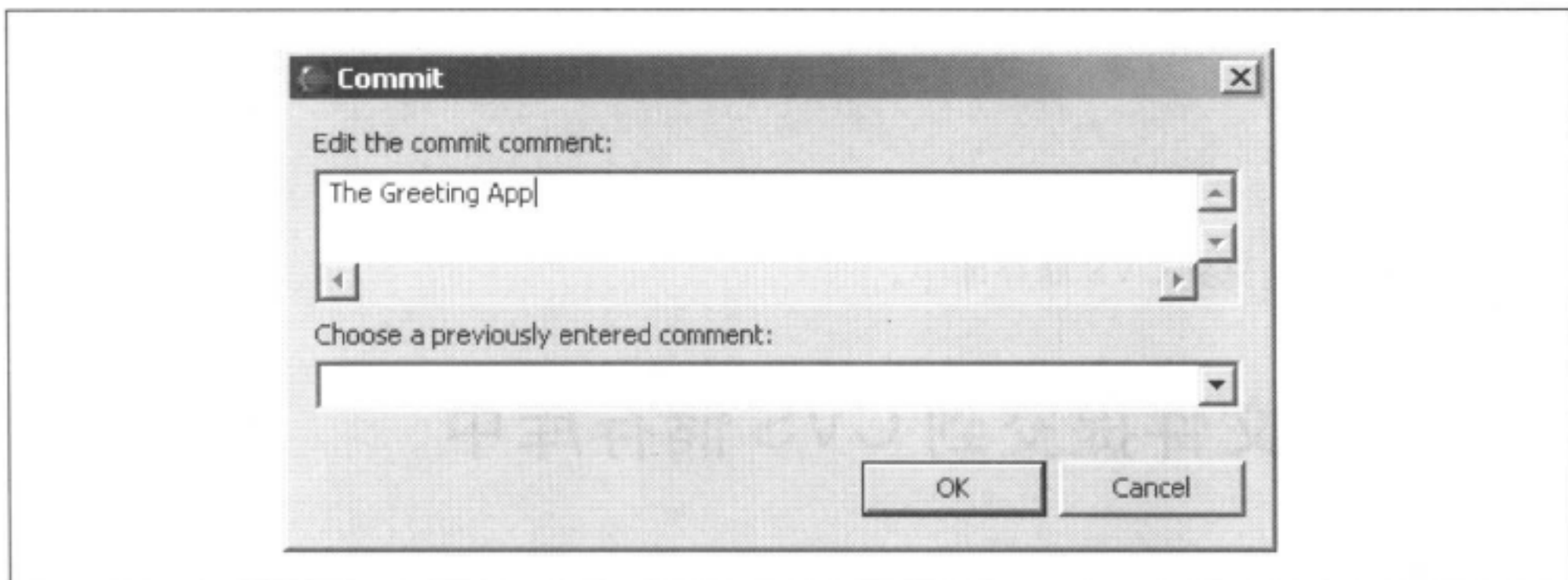


图 6-6：提交文件

也可以简单地右击一个文件，并选择 Team → Commit。如果该文件尚未处于版本控制之中，Eclipse 将询问是否要添加此文件；单击 Yes。Eclipse 将显示同一个 Commit 对话框，如图 6-6 所示，以便在提交文件之前输入文件的注释。

当提交文件时，它被上传到 CVS 储存库中，并得到一个版本号。如果需要的话，Eclipse 还可以为版本控制下的文件添加特殊的修饰；参阅 6.6 节。

参考

6.6 节，以可视化方式标记版本控制下的文件；*Eclipse* (O'Reilly) 一书的第 4 章。

6.6 以可视化方式标记版本控制下的文件问题

你想查看一下项目中的哪些文件处于版本控制之中。

解决方案

通过以下操作打开 CVS 修饰：选择 Window → Preferences → Workbench → Label Decorations，选中 CVS 复选框，并单击 OK。

讨论

要使 Eclipse 指示哪些文件处于版本控制之中，应打开 CVS 标签修饰功能。选择 Window → Preferences → Workbench → Label Decorations，打开 Preferences 对话框，然后选中 CVS 复选框，并单击 OK。这将使 Eclipse 在版本控制下的文件图标中添加一个金黄色的圆柱形标记，如图 6-7 中 *GreetingApp* 项目所示。注意，储存库中的文件还将获得一个 CVS 版本号，本例中为 1.1 版。

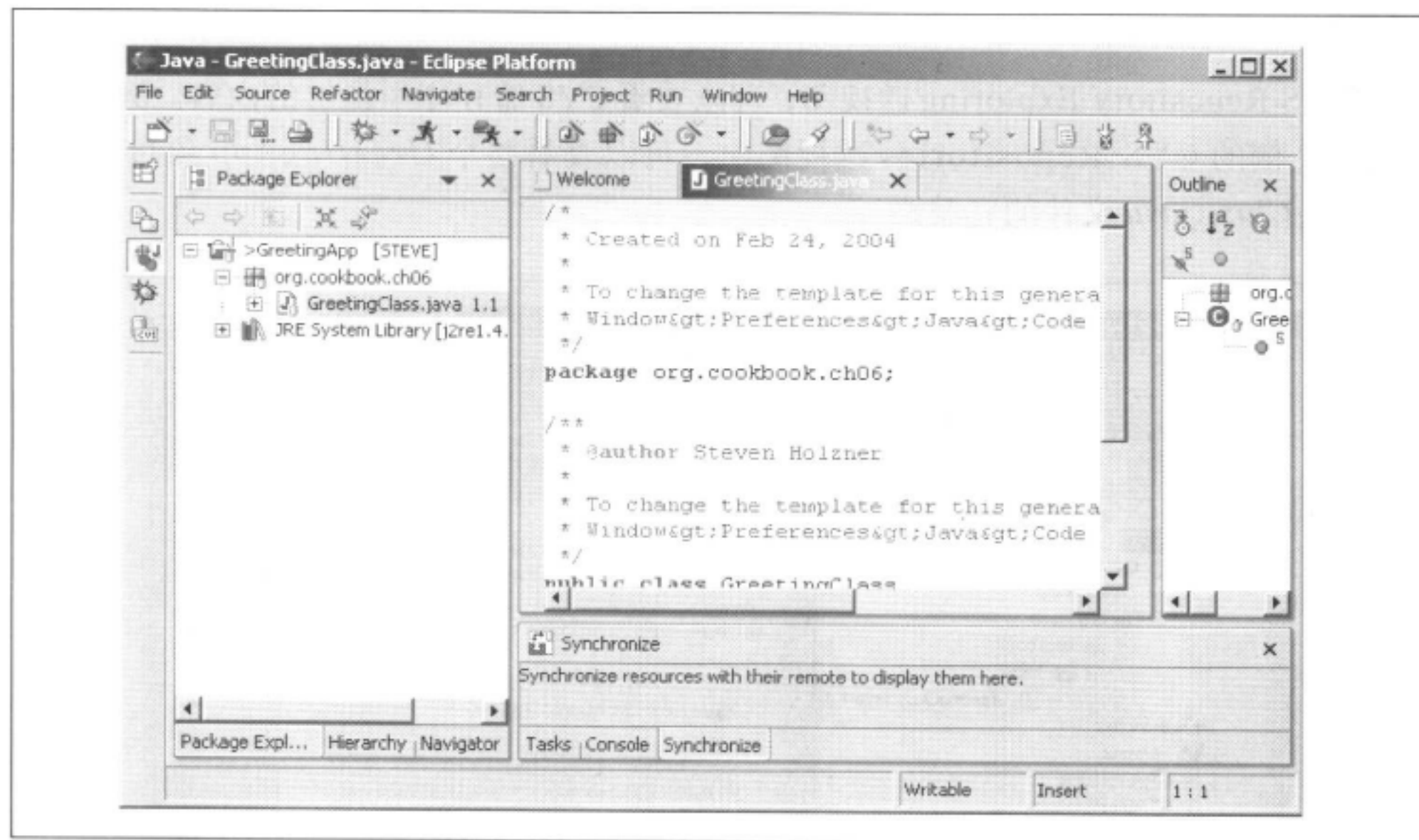


图 6-7：处于版本控制下的文件

参考

Eclipse (O'Reilly) 一书的第 4 章。

6.7 检查 CVS 储存库

问题

你想从 Eclipse 内部检查 CVS 储存库。

解决方案

使用 CVS Repository Exploring 透视图。选择 Window → Open Perspective → Other → CVS Repository Exploring, 打开此透视图。如果该透视图之前已经打开, 说明 Eclipse 应该已经将其添加到 Open Perspective 菜单中, 请选择 Window → Open Perspective → CVS Repository Exploring。

讨论

通过 CVS Repository Exploring 透视图, 可以查看 CVS 储存库中能够的内容。例如, 在图 6-8 左侧的 CVS Repositories 透视图, 可以查看整个 *GreetingApp* 项目以及至 *GreetingClass.java* 文件的完整路径。

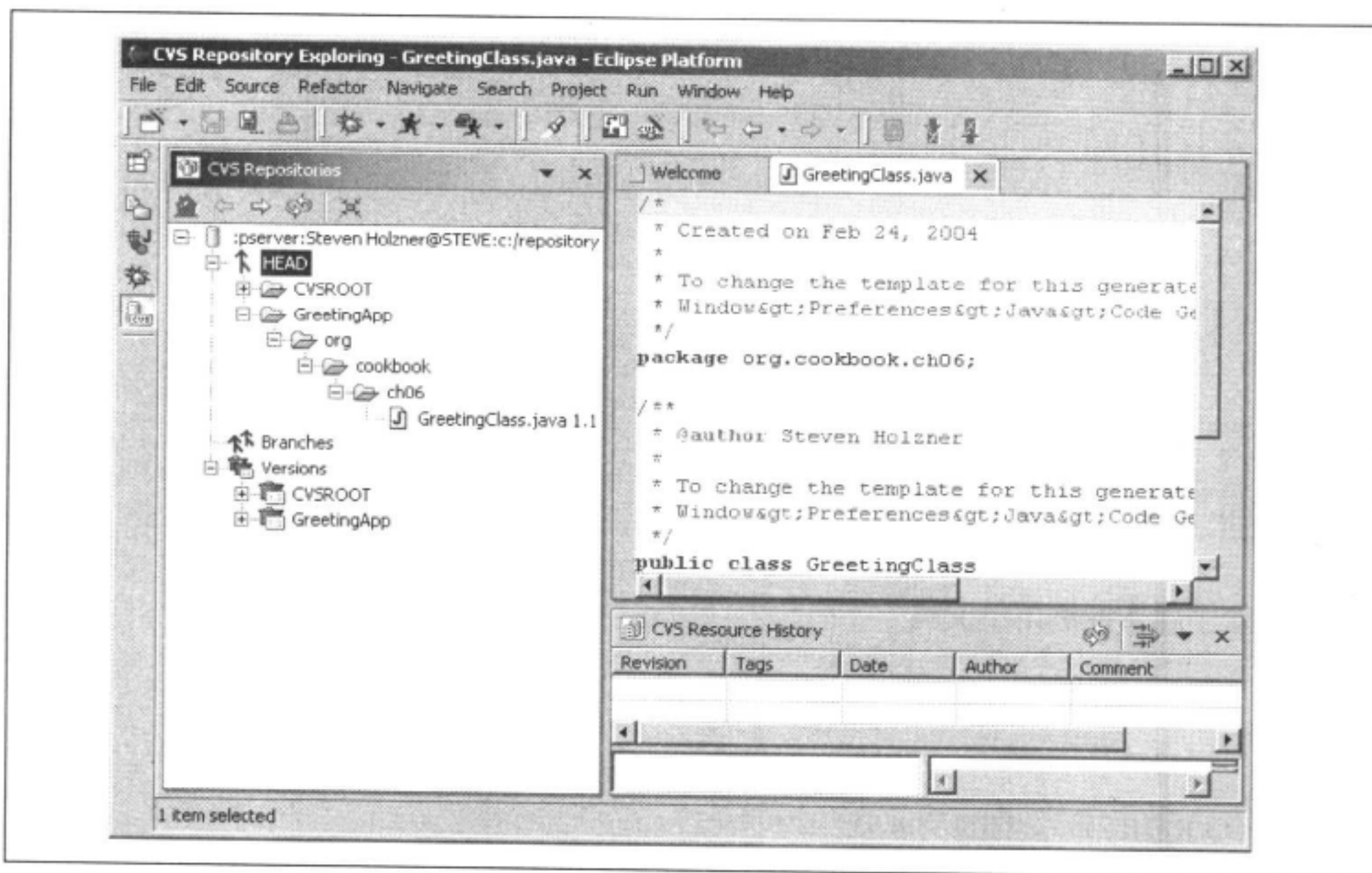


图 6-8: CVS Repository 透视图

项目 *GreetingApp* 位于储存库的 HEAD 节中, HEAD 节是主开发流。

注意: 还应注意 *CVSROOT* 目录, 其中存储着 CVS 的管理数据; Branches 节点存储着其他开发分支的所有文件; 而 Versions 节点显然存储着标记的版本。

Eclipse 3.0

在 Eclipse 3.0 中，可以确定谁应该对错误负责。右击一个文件，并选择 Team → Show Annotation，打开 CVS Annotation 视图，其中显示了在 CVS Repositories 视图中选定的文件。当在编辑器中选择一行代码时，CVS Annotation 视图可以揭示是谁将其发布到文件中的。

参考

6.8 节，从 CVS 储存库检出项目；6.13 节，创建 CVS 分支；*Eclipse* (O'Reilly) 一书的第 4 章。

6.8 从 CVS 储存库检出项目

问题

有人想从 CVS 储存库中检出你开发的一个项目。

解决方案

在 Eclipse，在 CVS Repositories 视图中右击一个文件，并单击 Check Out As。然后使用 Check Out 对话框检出该项目。

讨论

如果其他人想检出你存储在 CVS 储存库中的一个模块，正如本章前面所述，他必须创建至储存库的连接。要创建连接，应该右击 CVS Repositories 视图，选择 New → Repository Location；然后输入 CVS 服务器的名称、储存库的位置、用户名、密码以及连接的类型。

然后打开 CVS Repositories 视图，查看储存库中的文件。要检出 *GreetingApp* 模块，可在 Repositories 视图中右击该模块，并从快捷菜单中选择 Check Out As。Eclipse 将打开 New Project 对话框，并自动创建一个与该 CVS 模块对应的新项目。

如果打算共享一个 Eclipse 项目，每个 CVS 模块都应该有其自己的 Eclipse *.project* 文件，可从 Repositories 视图的快捷菜单中选择 Check Out As Project，检出一个 Eclipse 项目文件，并将其添加到 Package Explorer 视图中。注意，如果代码不在 Eclipse 能够识别的某种项目中，Eclipse 将要求提供要创建的项目类型。

参考

Eclipse (O'Reilly) 一书的第4章。

6.9 从 CVS 储存库更新本地代码

问题

你想用 CVS 储存库中的代码更新本地代码。

解决方案

右击该文件，并选择 Team → Update，然后解决冲突（如果有冲突的话）。另外，如果你只想用 CVS 储存库中的代码替换本地版本，可右击该文件，并选择 Replace With → Latest From HEAD。

讨论

例如，假设有人想检出你的代码，并把下面这一行

```
public static void main(String[] args)
{
    System.out.println("No problem.");
}
```

修改为：

```
public static void main(String[] args)
{
    System.out.println("No problems at all.");
}
```

当她用其自己的 Eclipse 版本进行上述修改时，一个 > 字符将出现在尚未提交的文件前面，如图 6-9 所示。

当她提交其修改时，CVS 储存库中的 *GreetingClass.java* 的最新版本由 1.1 改为 1.2，如图 6-10 所示。

为了用储存库中的代码的最新版本（现在是 1.2 版，有其他开发人员存储在储存库中）更新你的本地代码，可在你的 Eclipse 版本中右击该项目或文件，并选择 Team → Update。这样，如果不存在冲突的话，你的版本的项目文件将升级到 1.2 版。

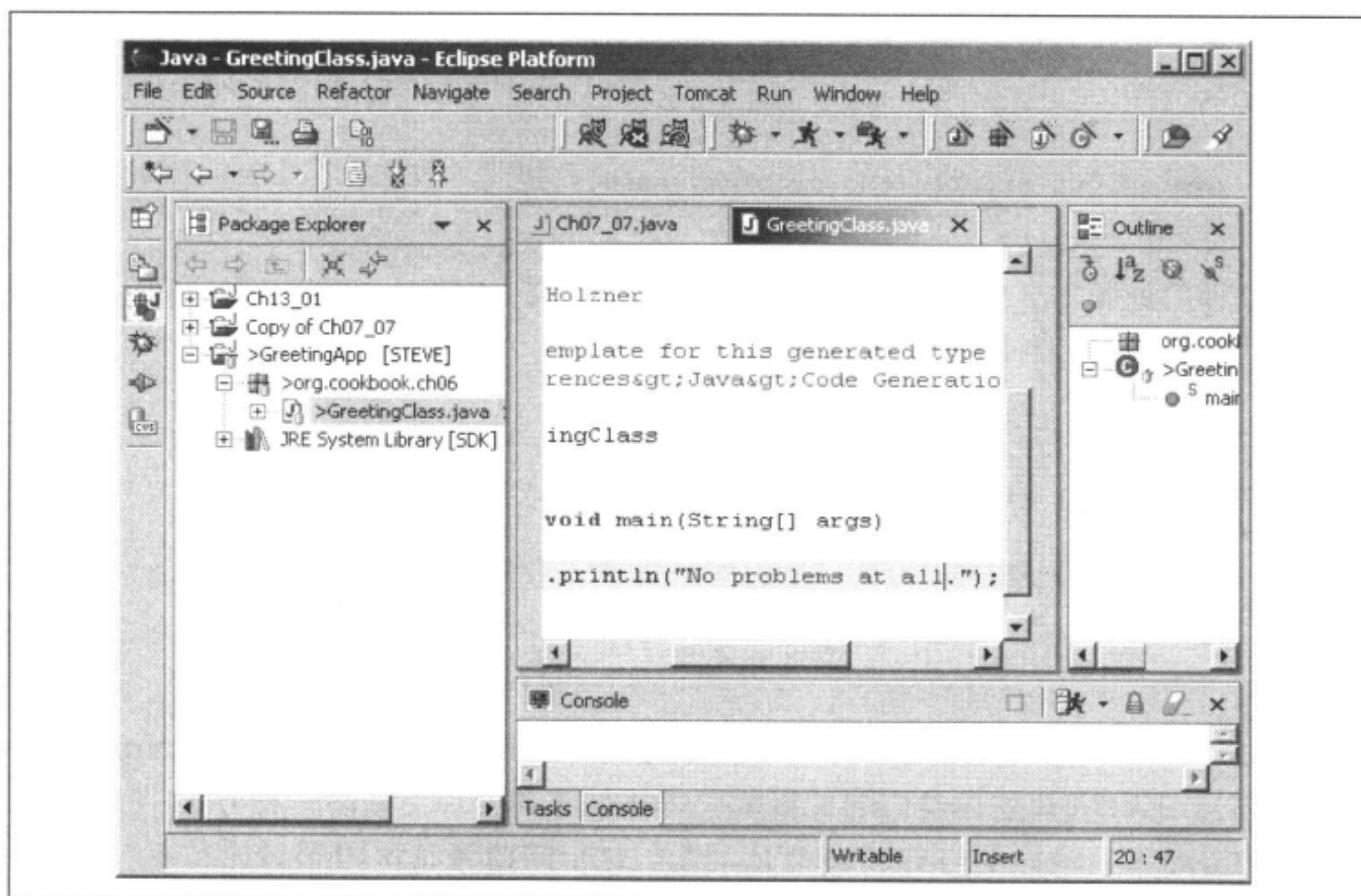


图 6-9: 输出的修改准备就绪

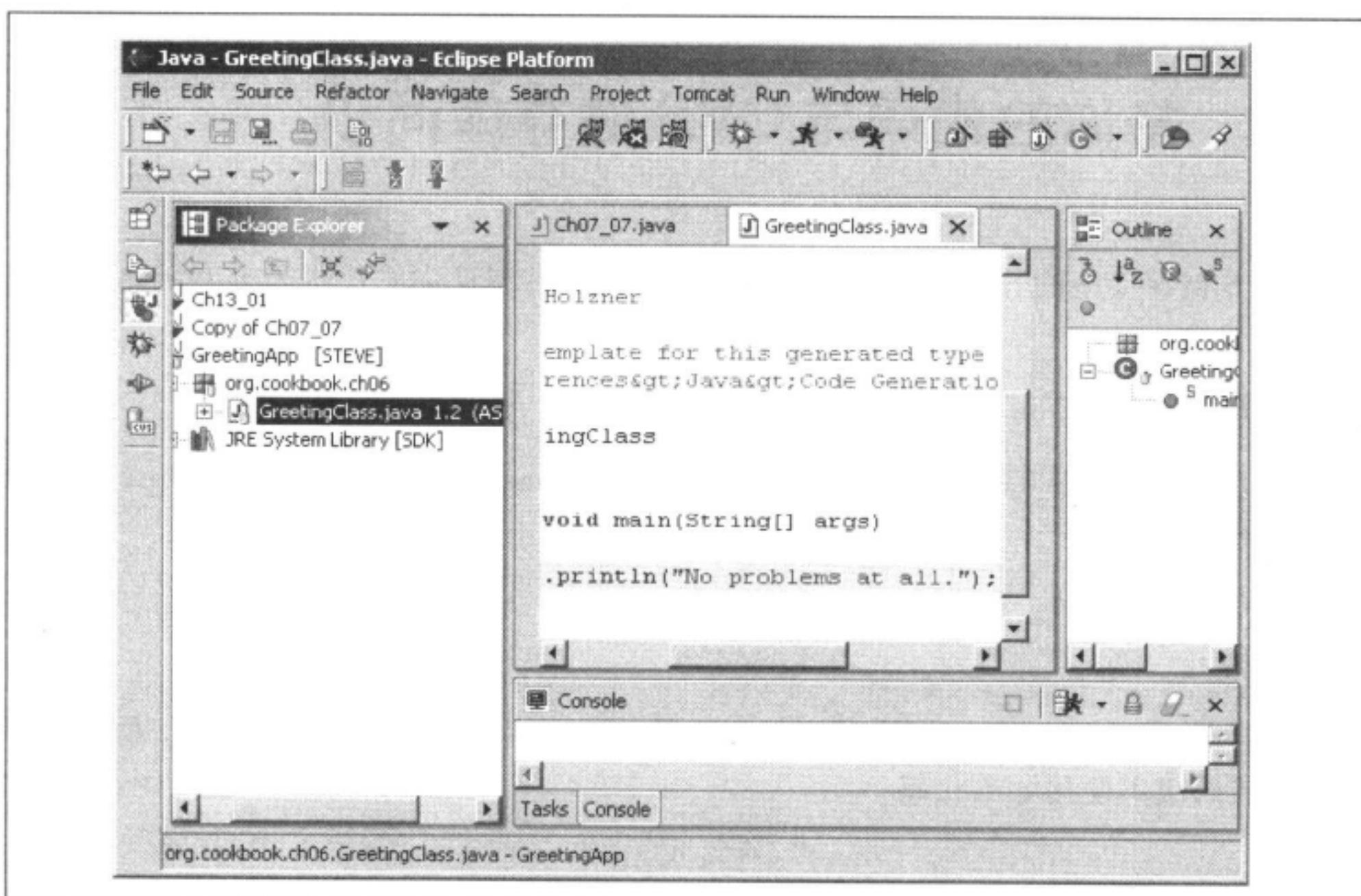


图 6-10: CVS 储存库中的新版本

如果你对这一行代码进行了不同的修改，比如改为：

```
public static void main(String[] args)
{
    System.out.println("No problems here.");
}
```

这将与文件的新版本（1.2 版）中的这一新行产生冲突。Eclipse 将标记该冲突，方法是在你的代码中列出两种版本，并添加某种 CVS 标记，如下所示：

```
public static void main(String[] args) {
<<<<<<< GreetingClass.java
    System.out.println("No problems here.");
=====
    System.out.println("No problems at all.");
>>>>>>> 1.2
}
```

你能够处理代码中存在的这些冲突（在 CVS 标记得解决并被删除之前，Eclipse 不会编译代码）。让 Eclipse 处理类似的更新是从 CVS 储存库处理更新的一种方式，但如果修改的地方非常多，最好是与 CVS 储存库同步，这个问题将在 6.10 节讨论。

注意，如果只想用 CVS 储存库中的项目主开发流中的文件的最新版本，替换该文件的本地版本，可右击该文件，并选择 Replace With → Latest From HEAD。

6.10 将你的代码与 CVS 储存库同步

问题

你对 CVS 储存库中的一个文件进行了大量的修改，并想使你的代码为大家所熟悉。

解决方案

右击该项目，并选择 Team → Synchronize with Repository。然后注意 Eclipse 并排显示的同步问题。

讨论

通过与储存库同步，可以以比合并更新更容易的方式对比所做的修改。例如，假设储存库中的代码版本使用如下代码：

```
public static void main(String[] args)
{
```

```
        System.out.println("No problems at all.");  
    }
```

而你将这一行代码修改为：

```
public static void main(String[] args)  
{  
    System.out.println("No problems here.");  
}
```

为了使代码与储存库中的文件版本同步，可右击该项目，并选择 Team → Synchronize with Repository。然后在 Structure Compare 视图中双击 *GreetingClass.java* 节点，以查看该文件的同步问题。其结果如图 6-11 所示。

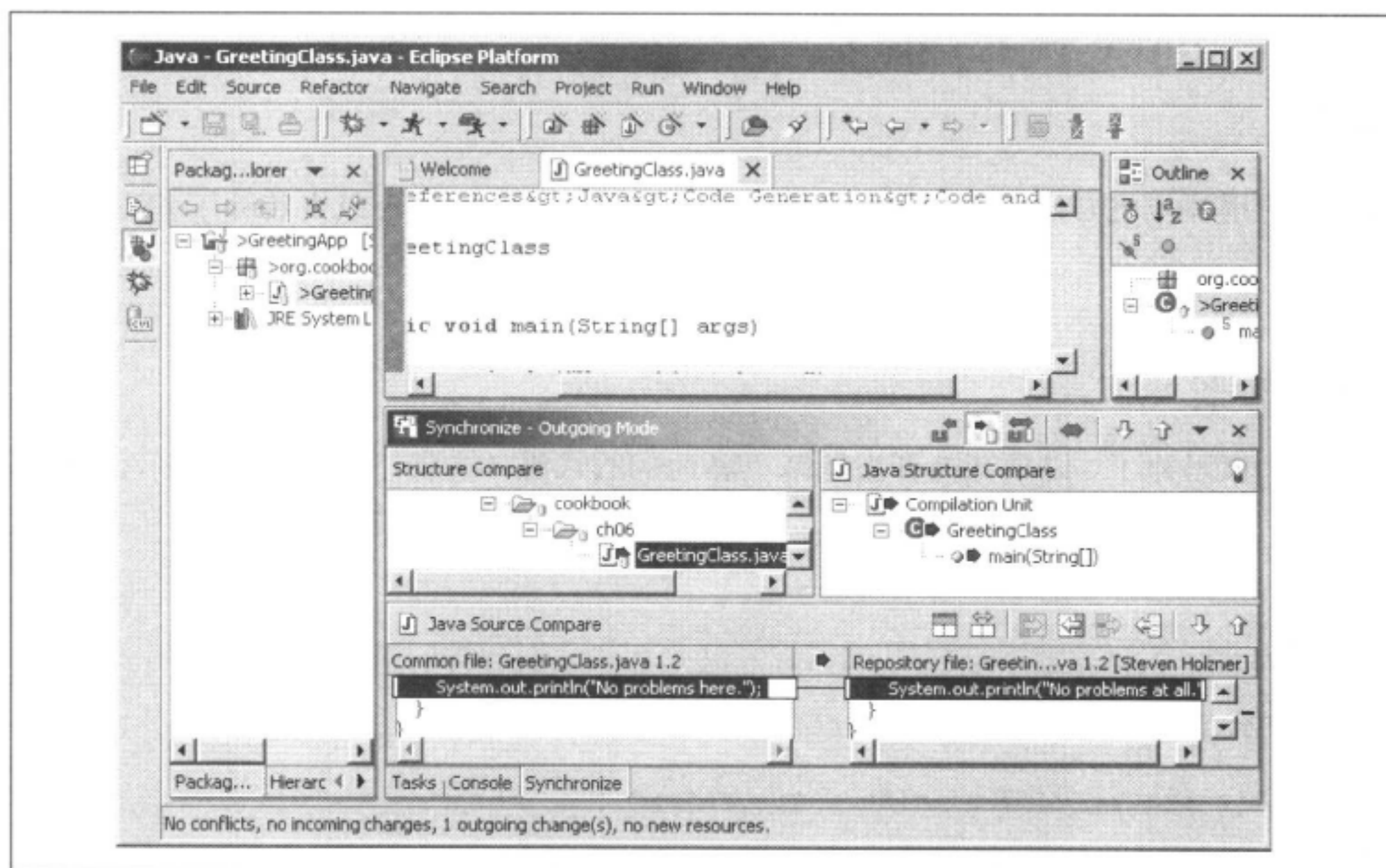


图 6-11：同步代码

注意，并排对比位于图 6-11 的底部，在此将本地文件与储存库中的文件进行比较。如图 6-11 所示，两个文件之间有差异的地方将出现连线。

使用 Java Source Compare 视图右侧的向上和向下箭头按钮，可以浏览所做的修改。可以使用导航按钮旁边的两个按钮接受修改或进行修改。左箭头按钮可以把储存库中的当前版本复制到本地代码中，而右箭头可以把本地代码中的最新修改复制到储存库中。在代码的本地版本与储存库中的代码同步之后，即可将修改提交到储存库中。

Eclipse 3.0

使用 Eclipse 3.0 可以很方便地查看文档的本地版本与 CVS 储存库中的文档版本之间的差异。右击 Quick Diff 标尺，并选择 Set QuickDiff Reference item to CVS Repository。这将使 QuickDiff 栏将最近的修改与 CVS 储存库中的版本进行对比——十分方便。

参考

4.6 节，根据本地历史记录比较文件；4.7 节，从本地历史记录恢复元素和文件；6.9 节，从 CVS 储存库更新本地代码；*Eclipse* (O'Reilly) 一书的第 4 章。

6.11 创建代码补丁

问题

你需要使用补丁将开发与另一个开发团队进行协调；他们可以安装补丁，以更新其代码。

解决方案

创建一个代码补丁，以便其他开发团队更新他们的代码（注意，这是代码补丁，而非二进制补丁；Eclipse 可以使用这种补丁更新源代码，以便与另一版本匹配）。

讨论

假设你的代码版本显示文本 "No problems here.":

```
public static void main(String[] args)
{
    System.out.println("No problems here.");
}
```

但另一个开发团队使用 CVS 储存库中的代码显示文本 "No problems at all.":

```
public static void main(String[] args)
{
    System.out.println("No problems at all.");
}
```

为了更新其他开发人员的代码，而不改变版本号，可以创建一个代码补丁。为了创建代码补丁，Eclipse 将本地代码与储存库中的代码进行比较，并创建一个补丁文件，用于保存其间的差异。

要创建代码补丁，以使用文件的本地版本更新储存库中的版本，可在本地保存文件，右击该文件，并选择 Team → Create Patch，打开如图 6-12 所示的对话框。

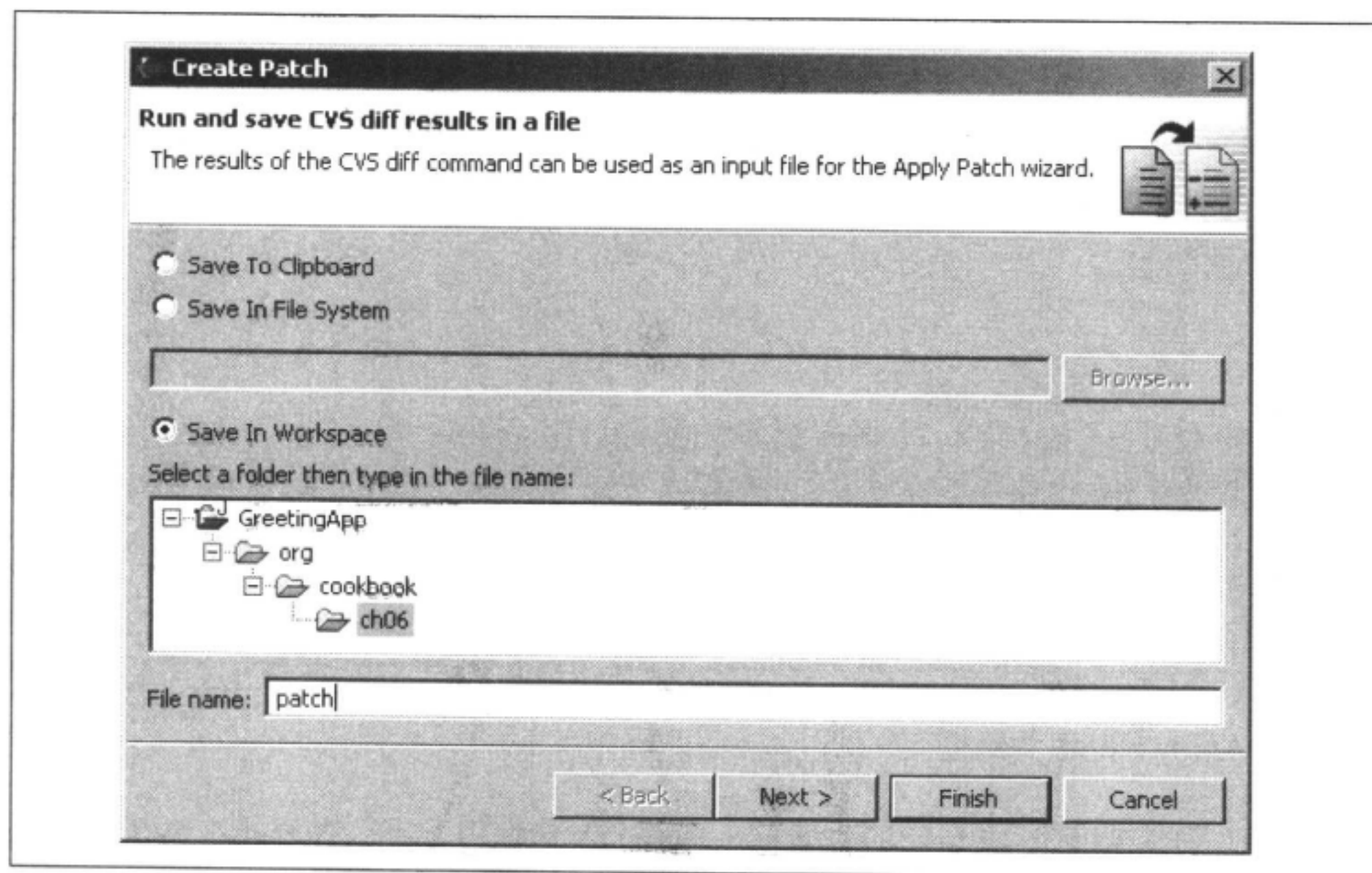


图 6-12：新建一个补丁

在这个例子中，我们把名为 *patch* 的补丁文件保存在当前工作区中，如图 6-12 所示。单击 Finish 按钮，保存补丁文件。

这将创建一个名为 *patch* 的文本文件，如下所示；可以看到，要删除的行用一个 - 号标记，而要添加的行用一个 + 号标记：

```
Index: GreetingClass.java
=====
RCS file: c:/repository/GreetingApp/org/cookbook/ch06/GreetingClass.java,v
retrieving revision 1.2
diff -u -r1.2 GreetingClass.java
--- GreetingClass.java      25 Feb 2004 16:34:07 -0000      1.2
+++ GreetingClass.java      25 Feb 2004 18:12:18 -0000
@@ -17,6 +17,6 @@

public static void main(String[] args)
{
-   System.out.println("No problems at all.");
+   System.out.println("No problems here.");
}
```


要在尚未打补丁的代码中应用新补丁，可在 Eclipse 中右击要更新的文件，并选择 Team → Apply Patch，打开如图 6-13 所示的对话框。

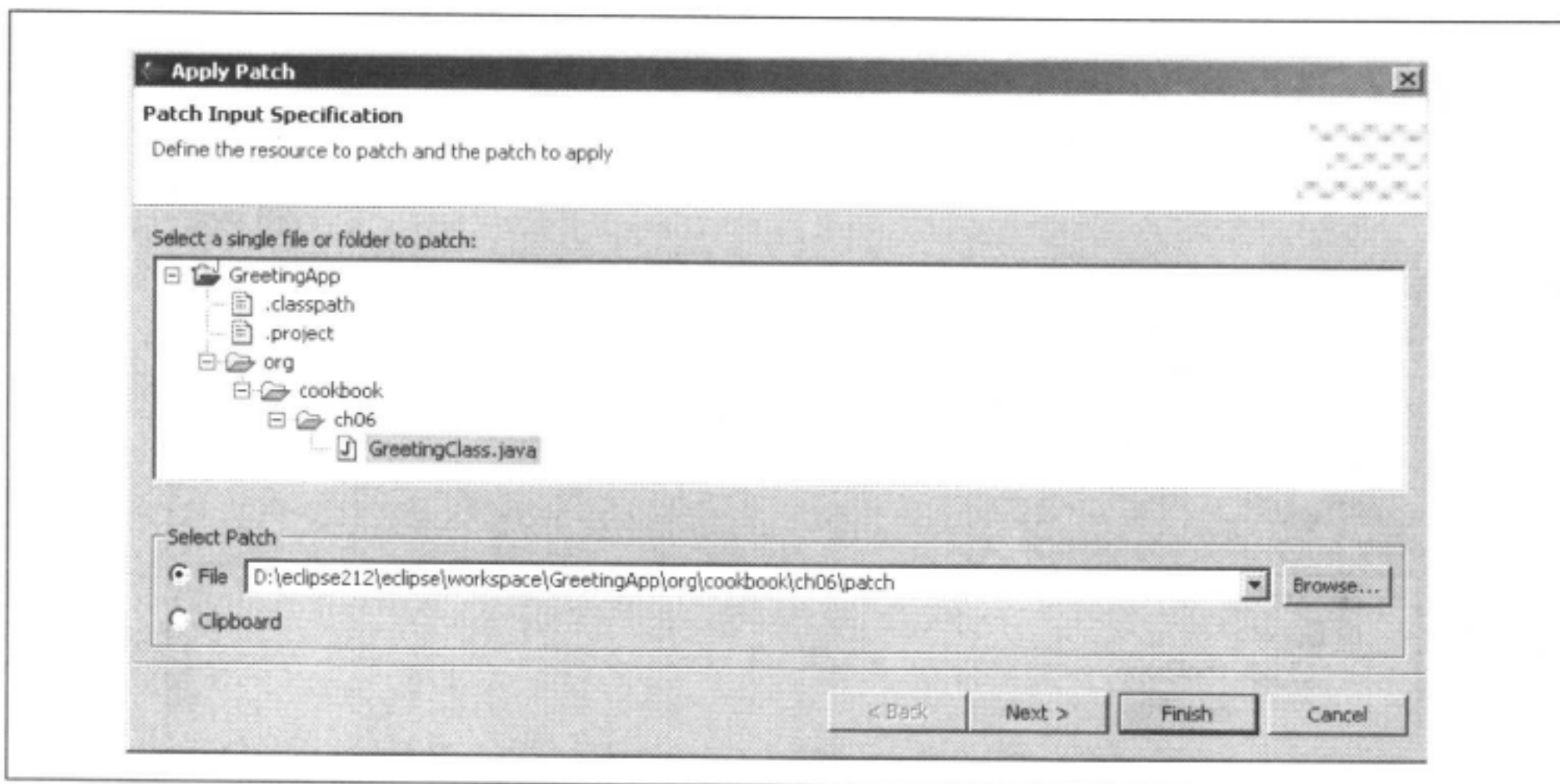


图 6-13: Apply Patch 对话框

单击 Next，打开如图 6-14 所示的对话框。在这个对话框中，可以查看补丁要在文件的本地版本中进行的修改。如图 6-14 所示，Eclipse 将把代码行：

```
System.out.println("No problems at all.");
```

修改为：

```
System.out.println("No problems here.");
```

要应用补丁，单击 Finish 按钮。

应用补丁，对其他团队的 Eclipse 安装中的代码进行上述修改，如图 6-15 所示。注意，文件的版本号没有改变，但文件用新代码进行了更新。

6.12 为代码版本命名问题

你获得了项目的一个里程碑版本，并且想按照名称将其保存在 CVS 储存库中，以便日后参考。

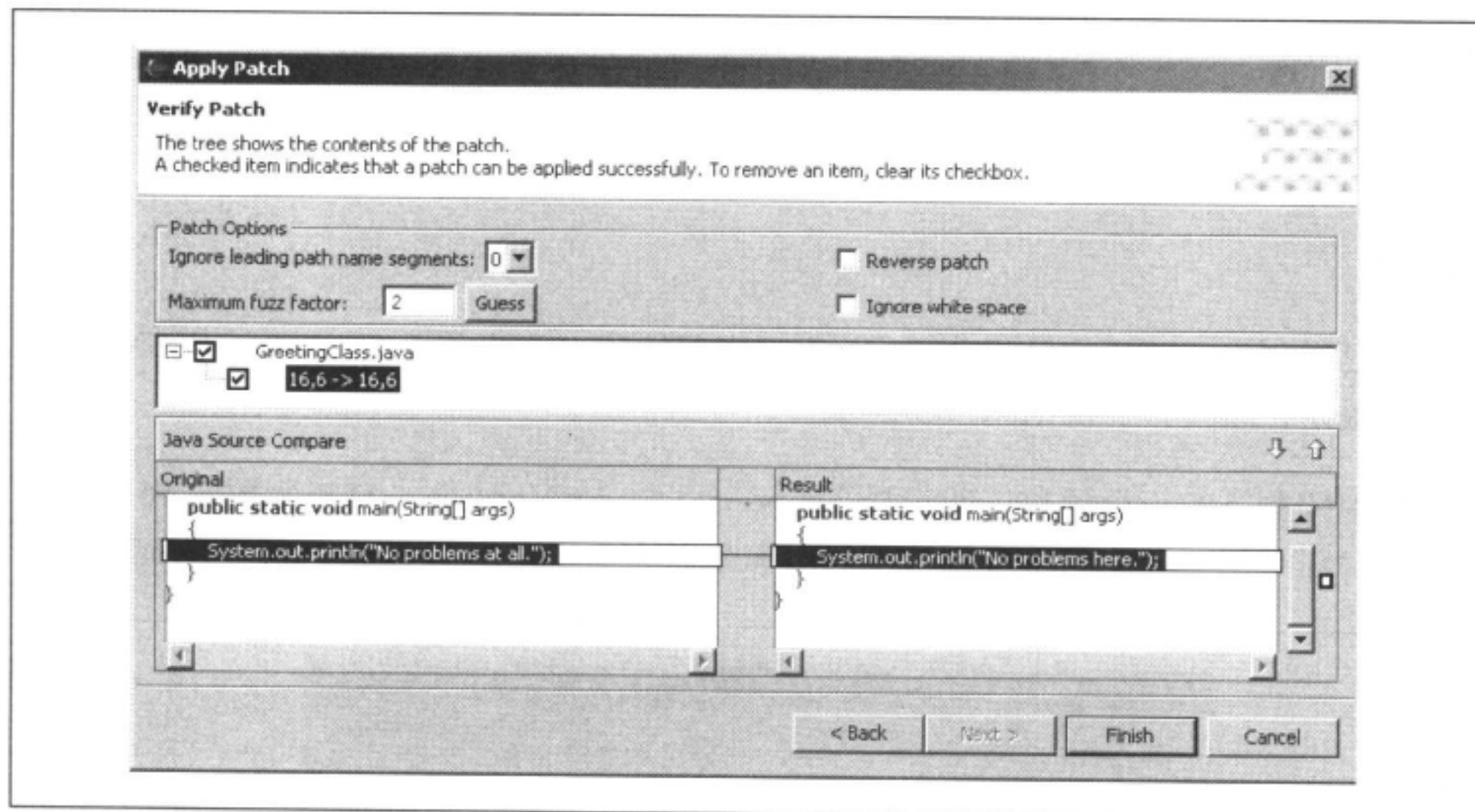


图 6-14: 接受补丁

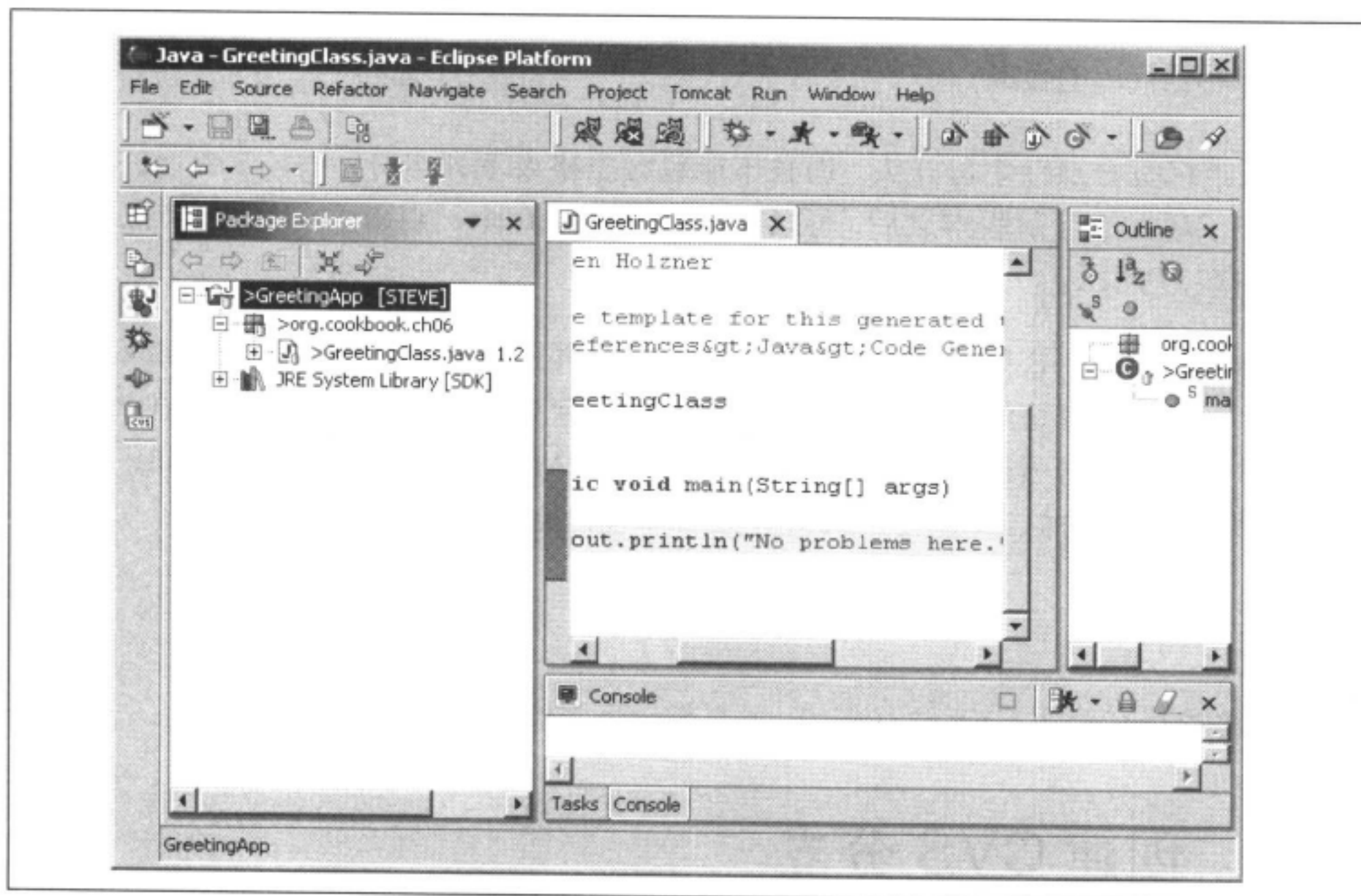


图 6-15: 应用代码补丁

解决方案

通过右击项目，并选择 Team → Tag As Version，用一个版本名称来标记项目。

讨论

如果你已经创建了项目的里程碑版本，你可能想按照名称将其保存。这样，使 CVS 存储标记的版本，以便日后根据名称访问项目。

右击版本控制下的一个项目，并选择 Team → Tag As Version，打开如图 6-16 所示的对话框。在本例中，我们将项目的当前版本命名为 Gold_Edition，如图 6-16 所示。

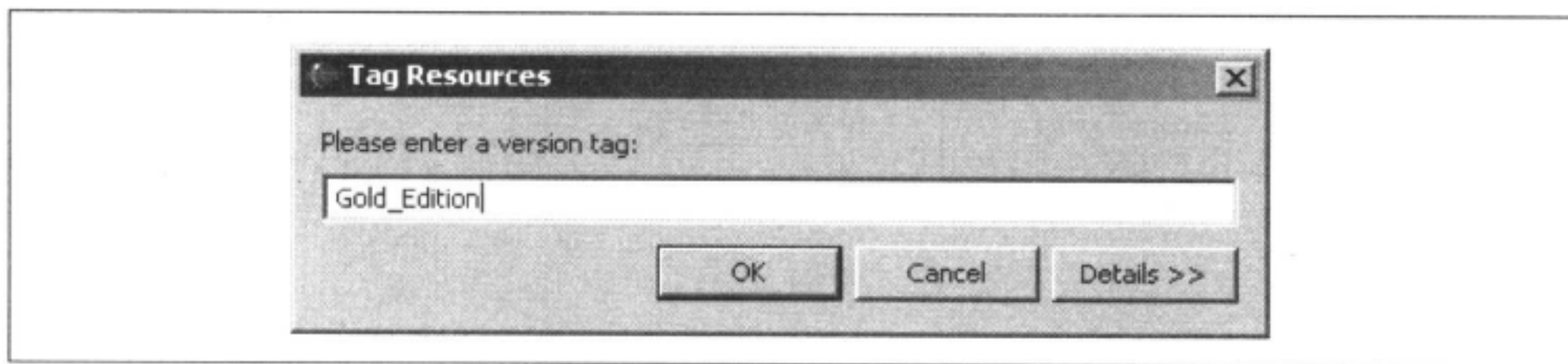


图 6-16：标记代码的版本

注意，版本标签必须以字母开头，而且不能包含空格或下列字符：`\$, . : ; @ | '。用这个名称标记了项目的当前版本之后，可以在 CVS Repositories 视图中 Versions 节点下找到它，如图 6-17 所示。

在 CVS Repositories 视图中右击标记的模块版本，并选择快捷菜单项，如 Check Out as Project，可以检出标记的模块版本，像使用其他任何 CVS 模块一样。还可以在 Package Explorer 视图中右击项目，并选择 Replace With → Another Branch or Version，打开如图 6-18 所示的对话框。选择要用来去替换当前项目的版本，并单击 OK。

参考

6.4 节，将项目存储在 CVS 储存库中；6.5 节，将文件提交到 CVS 储存库中。

6.13 创建 CVS 分支

问题

你想通过在开发树中新建一个分支，来开发代码的新版本，如测试版。

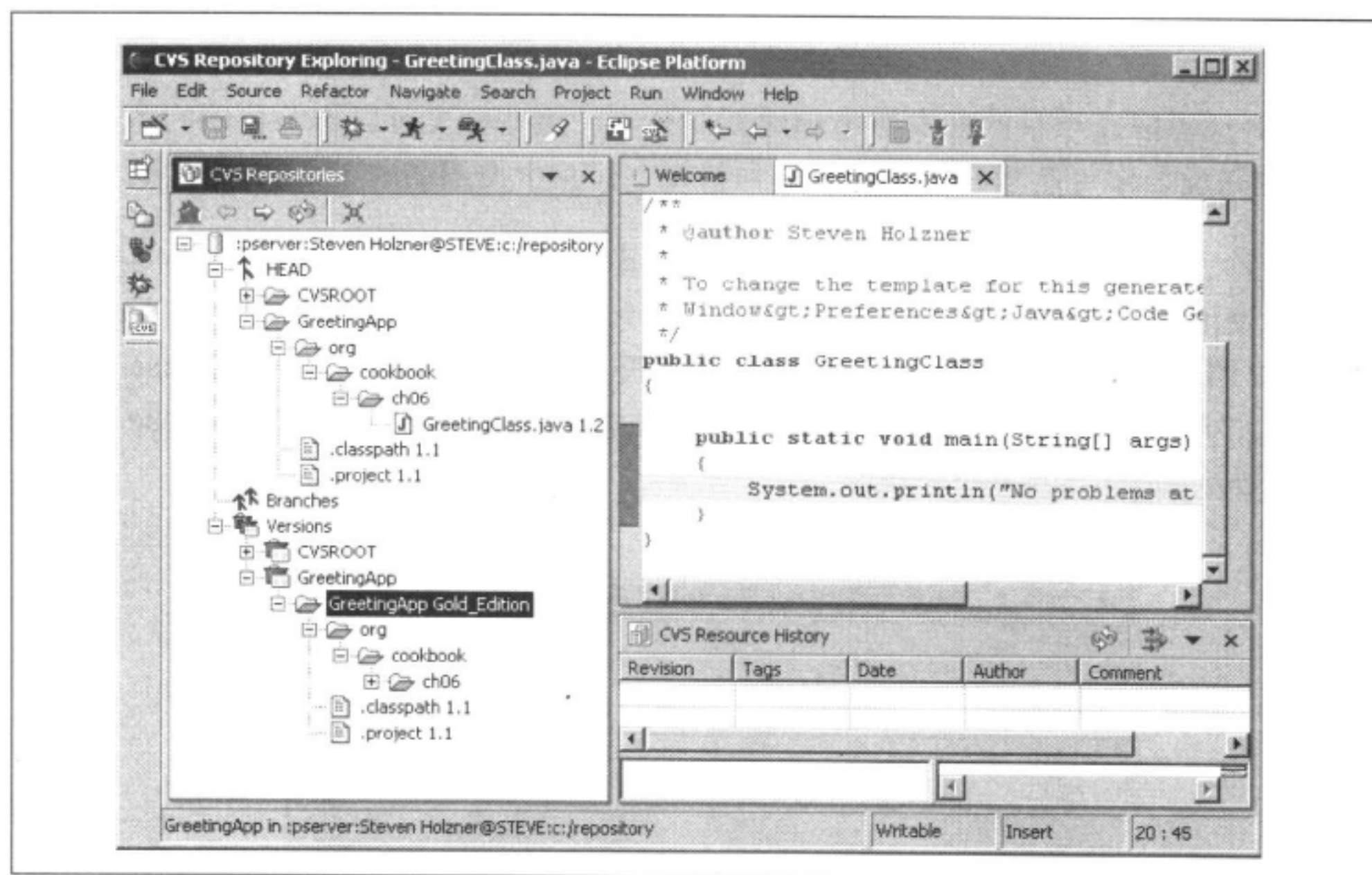


图 6-17：一个新标记的版本

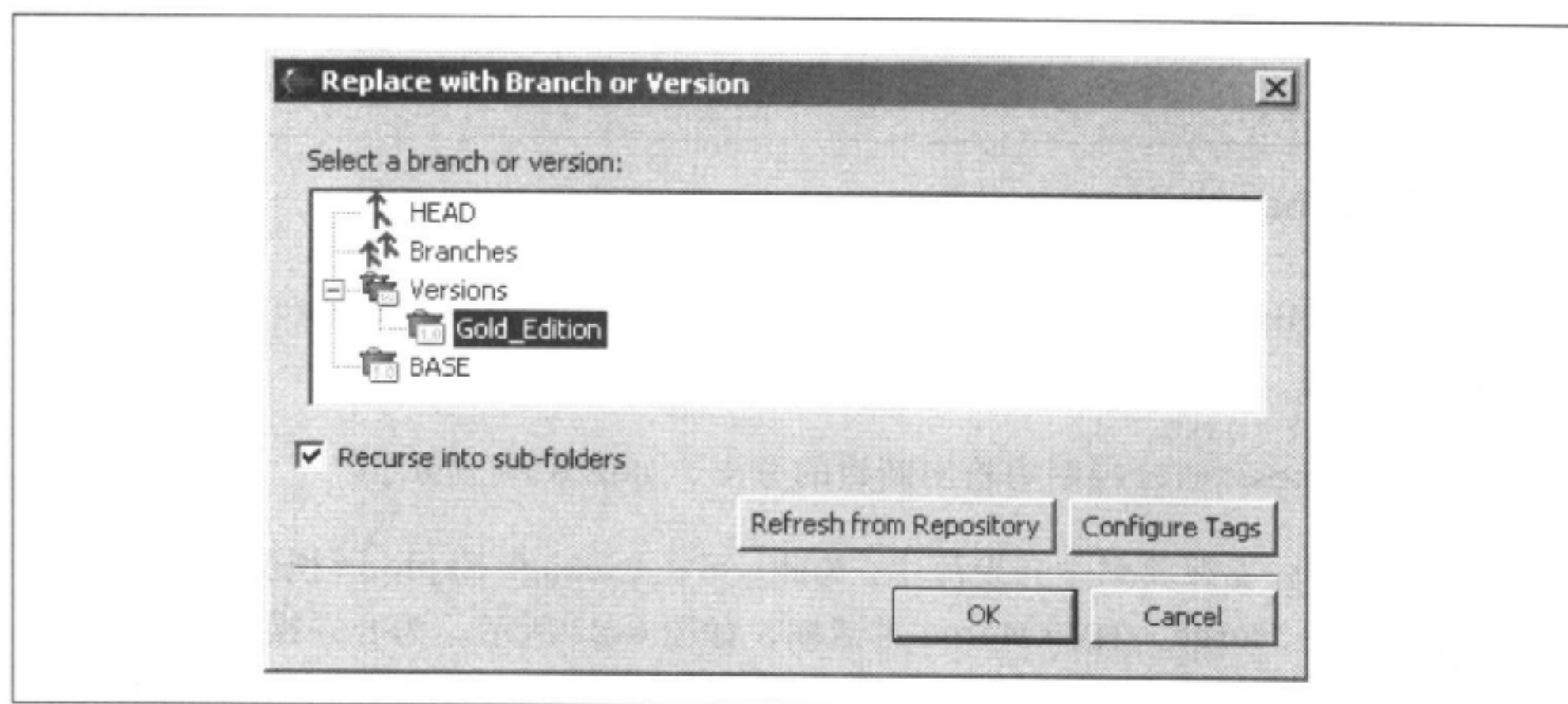


图 6-18：访问一个里程碑版本

解决方案

选择 Team → Branch，在项目的开发树中添加一个新的分支。

讨论

利用 CVS 还可以在项目的开发树中创建新的分支。这些分支可以作为代码的备用开发流；举例来说，你可能想开发代码的一种新版本，其中使用另一种语言的提示。

要创建一个分支，可右击项目，并选择 Team → Branch，打开 Create a new CVS Branch 对话框，如图 6-19 所示。在本例中，我们将这个新的分支命名为 Spanish_Version，如图 6-19 所示。同时，还可以为代码创建新的版本名称，作为参考，为 Eclipse 提供一个参考点，用于将分支合并到主开发流中（如果需要合并的话）；这里，Eclipse 建议的名称为 Root_Spanish_Version。

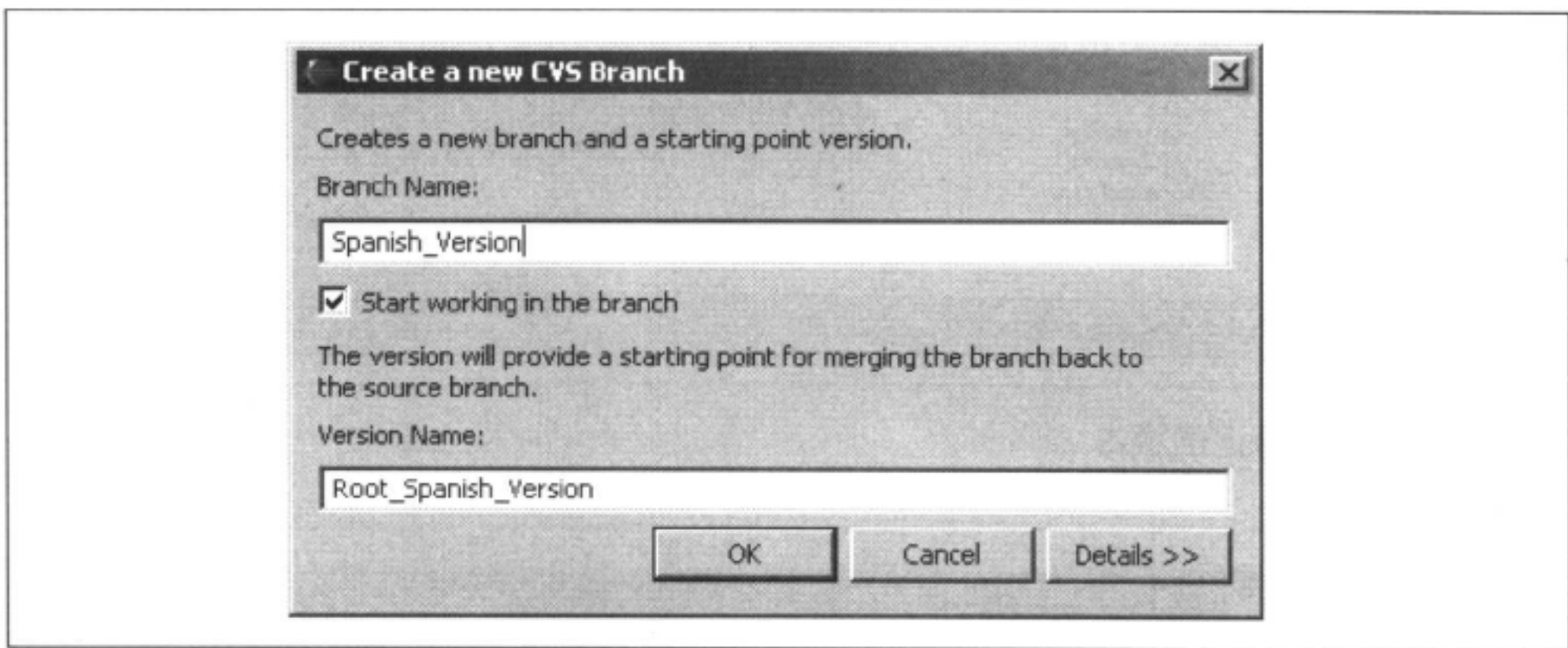


图 6-19：创建 Spanish_Version 分支

单击 OK，在 Eclipse 中打开新建的分支，如图 6-20 所示（注意 Package Explorer 视图中的项目名称）。

可以从 CVS Repositories 视图中检出新建的分支，如图 6-21 所示。

必要时，可以将分支合并到主开发流中。为此，可在 Package Explorer 视图中右击分支，并选择 Team → Merge，打开 Merge 对话框，如图 6-22 所示。为合并操作选择合并点——本例中为 Root_Spanish_Version——并单击 Next。

在下一个对话框中，选择要合并的起始分支。在本例中，选择 Spanish_Version，如图 6-23 所示。

在该对话框中，单击 Finish，完成合并操作。

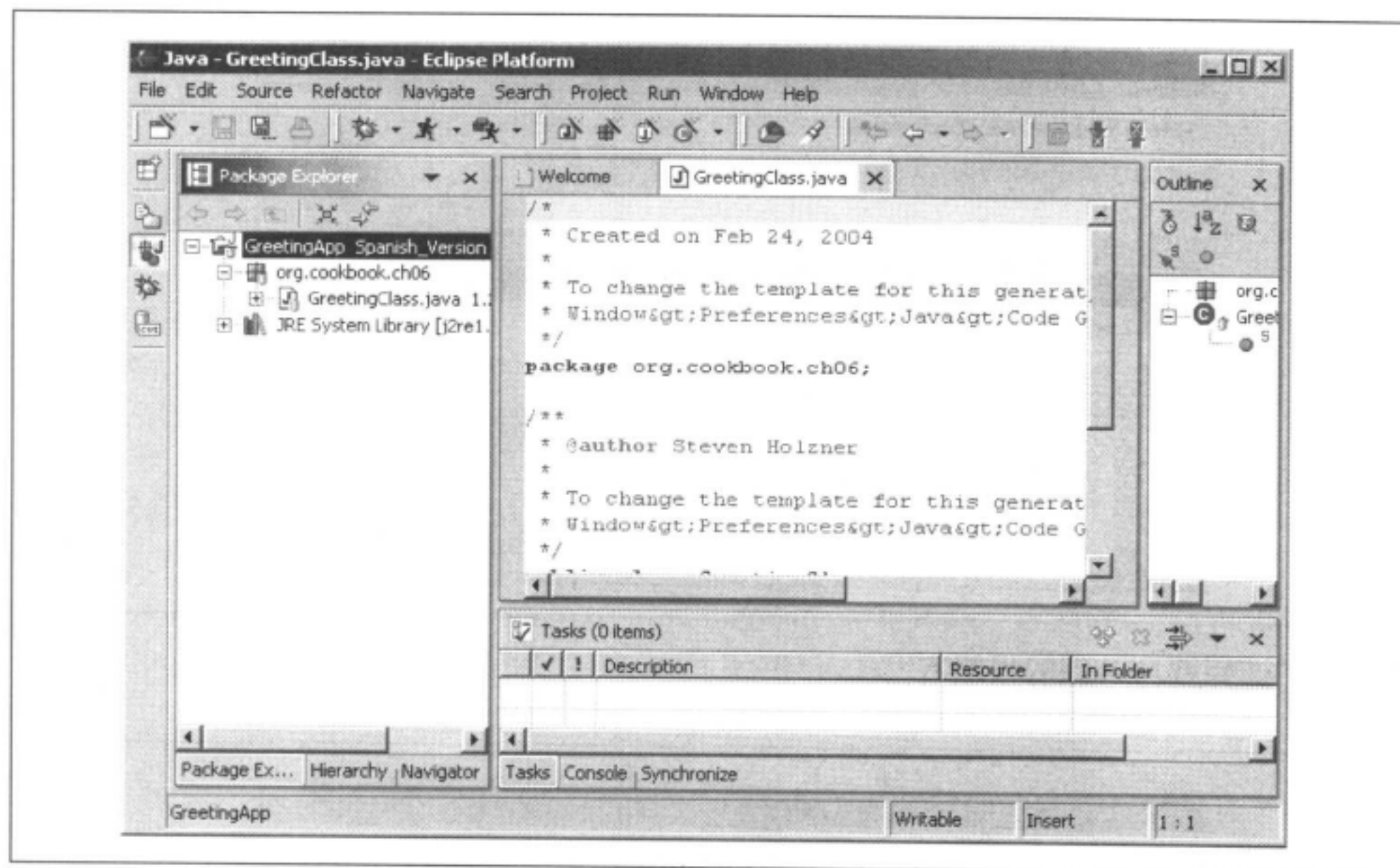


图 6-20：一个新的分支

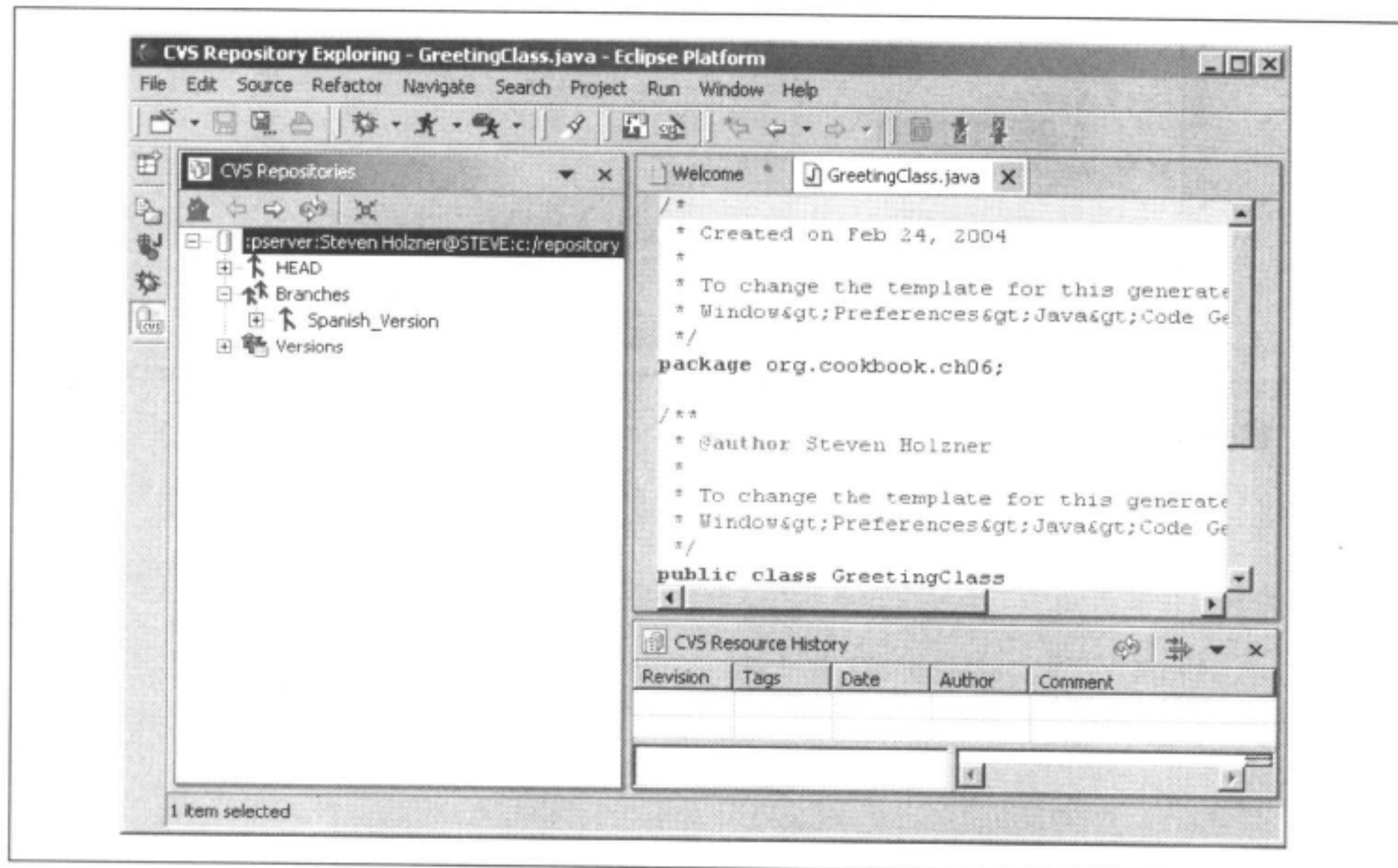


图 6-21：CVS Repositories 视图中的新建分支

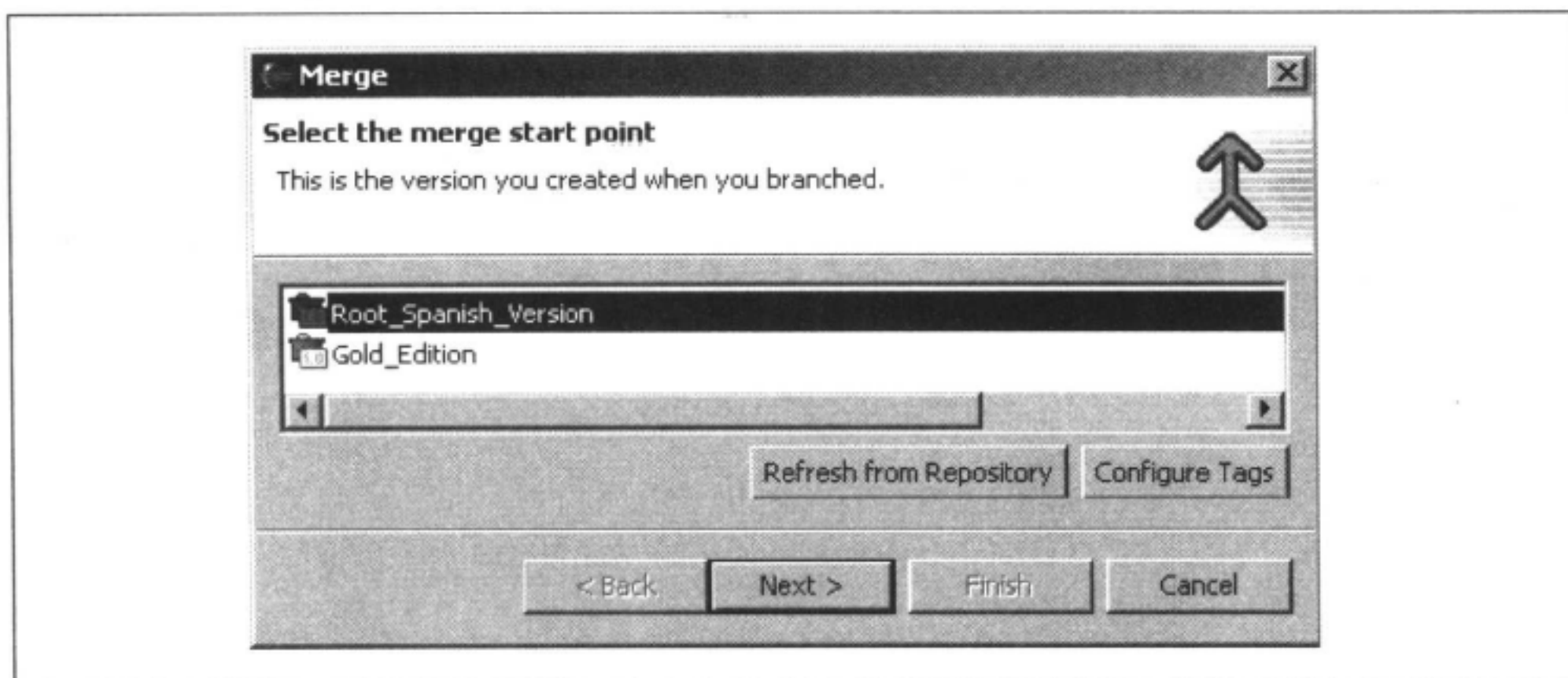


图 6-22：选择要合并到的根版本

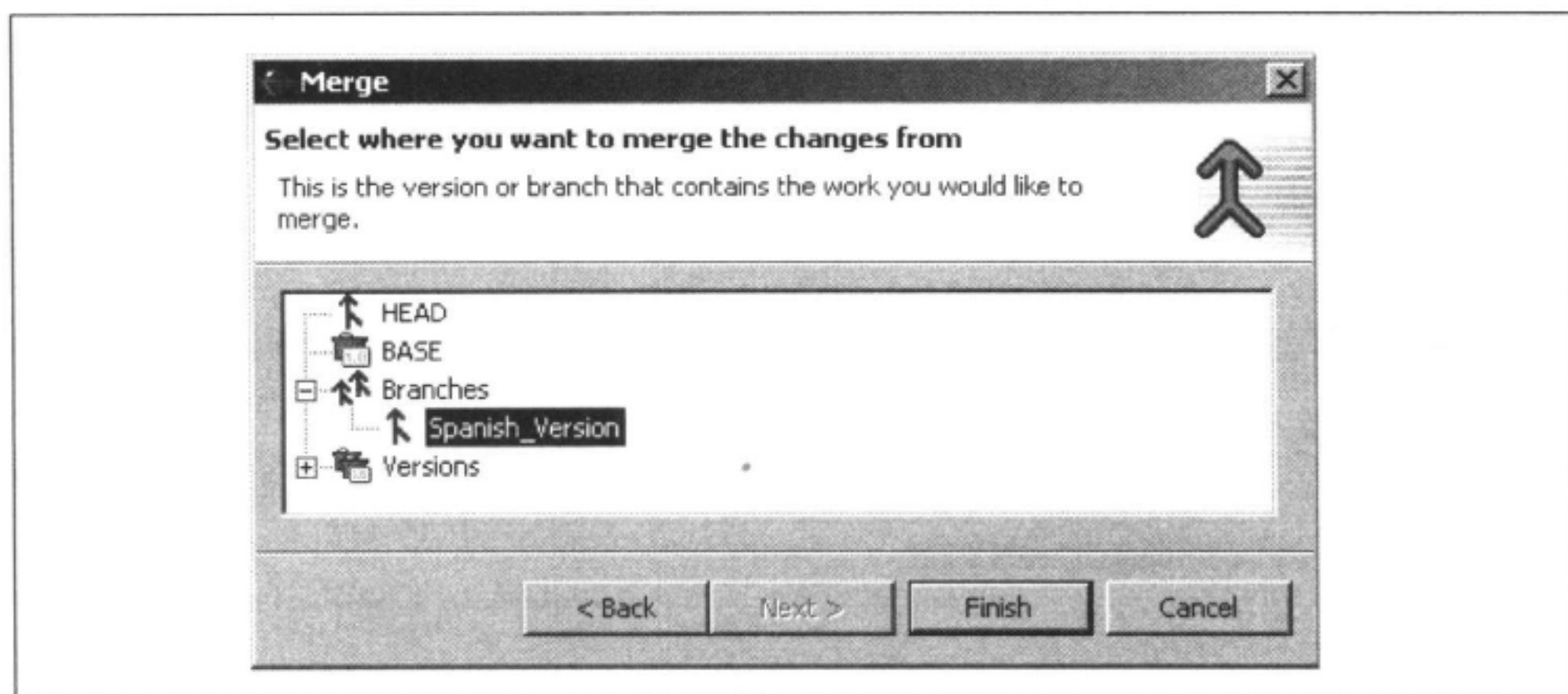


图 6-23：合并一个分支

Eclipse 和 Ant

7.0 简介

Eclipse 是一个非常好的 Java 集成开发环境，在编辑、调试、共享、编译和运行代码等方面表现优异。然而，先进的项目对开发过程有更高的要求。例如，可能需要编译多个文件，把编译过的文件复制到特定位置，创建 `.jar` 文件，删除以前版本的文件，等等。

所有这些任务都可以用著名的来自 Apache 的开发工具 Ant 来完成（参见站点 <http://ant.apache.org/>）。Ant 内置有非常强大的功能。它可以使开发过程自动化——编译、移动、复制和删除文件，创建和删除目录，等等。而现在，Ant 已经内置到了 Eclipse 中。如果你的项目需要的不仅仅是编译一二个文件，那么一定要阅读本章。

注意：本章讨论如何从 Eclipse 中使用 Ant，但并不打算详细讨论 Ant，这方面的内容本身就需要一整本书的篇幅。要详细了解如何使用这个开发工具，请参阅 *Ant: The Definitive Guide* (O'Reilly) 一书。

7.1 将 Ant 连接至 Eclipse

问题

你想开始从 Eclipse 中使用 Ant，但需要知道如何将 Ant 连接至 Eclipse。

解决方案

你所需要做的就是将一个 `build.xml` 文件添加到 Eclipse 项目中。Eclipse 知道，具有该名称的文件应作为一个 Ant 编译文件（build file）来处理。

讨论

作为示例，我们将创建一个 Ant 编译文件，*build.xml*，用它来编译一个 Eclipse 项目，将生成的 *.class* 文件存储在 *.jar* 文件中，而 *.jar* 文件存储在指定的目录中。按照屏幕提示进行操作，创建一个名为 *AntProject* 的 Eclipse。

把该项目的源文件存储在一个名为 *src* 的文件夹中，而项目的输出存储在一个名为 *bin* 的文件夹中。在 New Project 对话框中的第三步，即 Java Setting 对话框中，单击 Source 标签，并单击 Add Folder 按钮。然后单击 Create New Folder 按钮，打开 New Folder 对话框。在 Folder name 文本框中输入 *src*，并单击 OK 两次。Eclipse 将询问：

Do you want to remove the project as source folder and update build output folder to 'AntProject/bin'? (你想删除作为源文件夹的项目并把编译输出文件夹更新为 AntProject/bin 吗?)

单击 Yes，得到如图 7-1 所示的结果。单击 Finish，完成项目的创建工作，项目中现在包括 *src* 和 *bin* 文件夹。

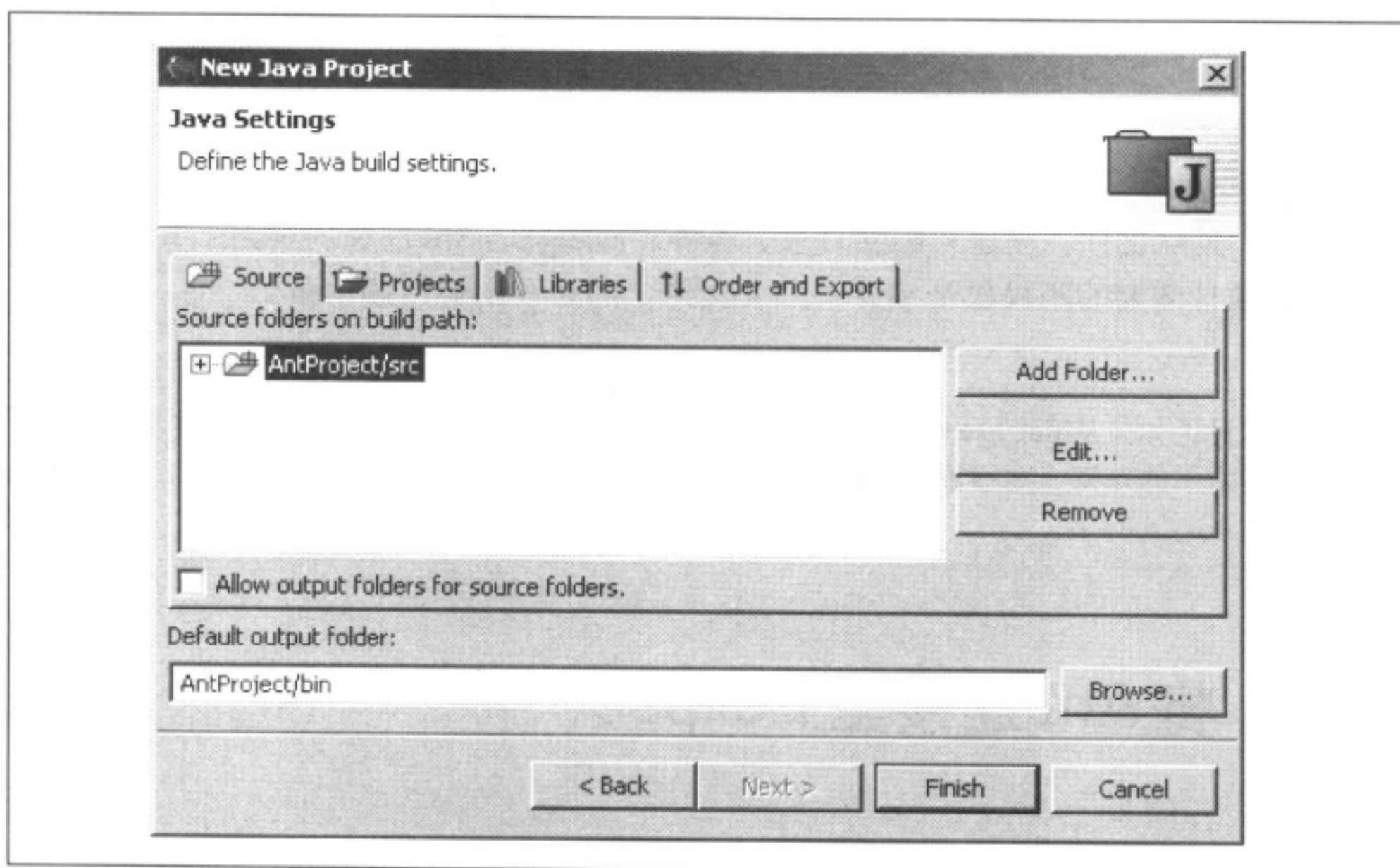


图 7-1：创建 AntProject 项目

在 *org.cookbook.ch07* 包中添加一个新的类，*AntClass*。我们只打算使用这个类中的一些示例代码来显示一条消息，如例 7-1 所示。

例 7-1: 一个简单的 Ant 测试类

```
package org.cookbook.ch07;

public class AntClass
{
    public static void main(String[] args)
    {
        System.out.println("This code is stored in a JAR file.");
    }
}
```

在 Package Explorer 视图中右击项目，并选择 New → File，在项目中添加一个名为 *build.xml* 的文件。在 File name 文本框中输入 *build.xml*，单击 Finish，这将创建文件 *build.xml*，并在 Ant 编辑器中打开它，如图 7-2 所示。注意，Eclipse 已经知道这是一个 Ant 编译文件，并在 Package Explorer 视图中为其提供一个蚂蚁图标。

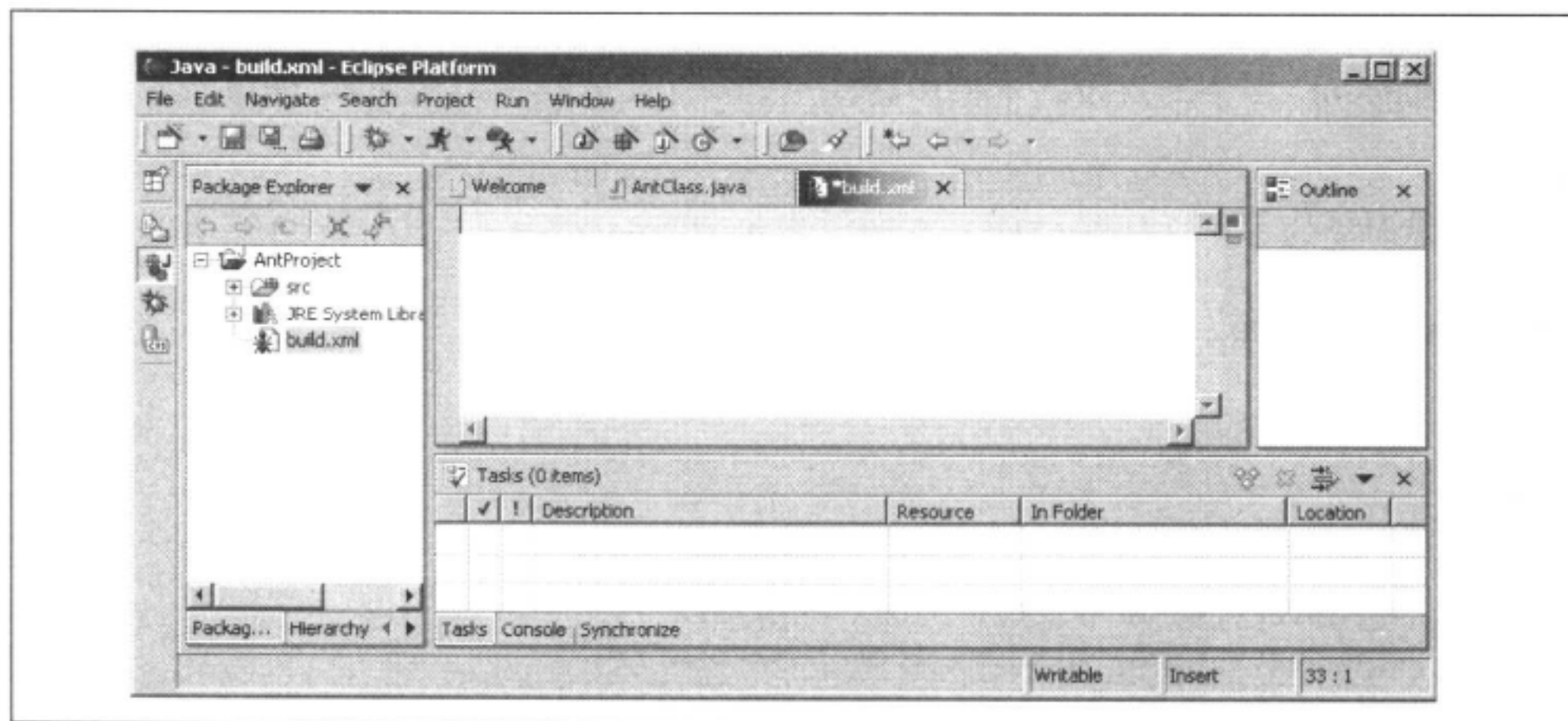


图 7-2: 创建 build.xml 文件

文件 *build.xml* 是一个 XML 文件，Ant 使用它来编译项目。文件的开头是标准的 XML 声明和一个 Ant 项目元素：

```
<?xml version="1.0" encoding = "UTF-8"?>
<project name="AntProject" default="Build" basedir=".">
    .
    .
    .
</project>
```

可以在 Ant 编译文件中设置属性，将以后要使用的术语定义集中在一个编译文件中，并与 Ant 交互。在本例中，我们通过将 *build.compiler* 属性设置为 *org.eclipse.jdt.core.JDTCompilerAdapter*，指定 Ant 使用与在 JDT 中相同的编译器，代码如下：

```
<?xml version="1.0" encoding = "UTF-8"?>
<project name="AntProject" default="Build" basedir=".>
  <property name="build.compiler"
    value="org.eclipse.jdt.core.JDTCompilerAdapter"/>
  .
  .
  .
</project>
```

另外，在编译文件中添加以下属性，这些属性与项目中使用的各种编译路径相对应：

srcDir

保存源代码的目录，本例中为 *src*。

binDir

二进制输出目录，本例中为 *bin*。

jarDir

创建的 *.jar* 文件的目录；本例中将使用 *bin* 目录下的一个名为 *lib* 的目录。

jarFile

将要创建的 *.jar* 文件的名称，本例中为 *AntProject.jar*。

在 Eclipse 编辑器中，将这些属性输入文件 *build.xml*：

```
<?xml version="1.0" encoding = "UTF-8"?>
<project name="AntProject" default="Build" basedir=".>
  <property name="build.compiler"
    value="org.eclipse.jdt.core.JDTCompilerAdapter"/>
  <property name="srcDir" location="src"/>
  <property name="binDir" location="bin"/>
  <property name="jarDir" location="${binDir}/lib"/>
  <property name="jarFile" location="${jarDir}/AntProject.jar"/>
  .
  .
  .
</project>
```

Ant 编译文件将根据 Ant 目标进行构建。在本例中，第一个目标是删除输出目录 *bin* 和 *.jar* 文件输出目录 *bin/lib* 中现有的所有内容，并重新构建这些目录。这一切都将在一个名为 *Initialization* 的目标中完成：

```
<?xml version="1.0" encoding = "UTF-8"?>
<project name="AntProject" default="Build" basedir=".>
  <property name="build.compiler"
    value="org.eclipse.jdt.core.JDTCompilerAdapter"/>
  <property name="srcDir" location="src"/>
  <property name="binDir" location="bin"/>
```

```

<property name="jarDir" location="${binDir}/lib"/>
<property name="jarFile" location="${jarDir}/AntProject.jar"/>

<target name="Initialization">
    <delete dir="${binDir}"/>
    <delete dir="${jarDir}"/>
    <mkdir dir="${binDir}"/>
    <mkdir dir="${jarDir}"/>
</target>
.
.
.
</project>

```

下一个 Ant 目标, *Compilation*, 将编译源文件, 并把生成的 *.class* 文件放在 *bin* 目录中。注意, 这个目标的实现依赖于 *Initialization* 目标已经成功实现:

```

<?xml version="1.0" encoding = "UTF-8"?>
<project name="AntProject" default="Build" basedir=".">

    <property name="build.compiler"
        value="org.eclipse.jdt.core.JDTCompilerAdapter"/>
    <property name="srcDir" location="src"/>
    <property name="binDir" location="bin"/>
    <property name="jarDir" location="${binDir}/lib"/>
    <property name="jarFile" location="${jarDir}/AntProject.jar"/>

    <target name="Initialization">
        <delete dir="${binDir}"/>
        <delete dir="${jarDir}"/>
        <mkdir dir="${binDir}"/>
        <mkdir dir="${jarDir}"/>
    </target>

    <target name="Compilation" depends="Initialization">
        <javac srcdir="${srcDir}"
            destdir="${binDir}"/>
        </javac>
    </target>
    .
    .
    .
</project>

```

Jar 目标把 *AntProject.class* 压缩到一个 *.jar* 文件中, 并把该文件存储为 *\${jarfile}*。注意, 这个目标的实现依赖于目标 *Initialization* 和 *Compilation* 的成功实现。

```

<target name="Jar" depends="Initialization, Compilation">
    <jar destfile="${jarFile}" basedir="${binDir}"/>
</target>

```

现在, 为这个编译文件创建主要目标, 即 *Build*。这个目标通过强制要求其他目标必须

完成,来协调所有其他目标,并显示一条消息“Ant is building your project.”。下列代码说明了 Build 目标的组成。

```
<target name="Build" depends="Initialization, Compilation, Jar">
    <echo message="Ant is building your project."/>
</target>
```

最后,使用<project>元素使 Build 目标成为编译文件的默认目标,以便 Ant 知道从该目标开始。通过例 7-2 中的完整的编译文件,可以了解如何进行设置。

例 7-2: 完整的 build.xml 文件

```
<?xml version="1.0" encoding = "UTF-8"?>
<project name="AntProject" default="Build" basedir=".">

    <property name="build.compiler"
        value="org.eclipse.jdt.core.JDTCompilerAdapter"/>
    <property name="srcDir" location="src"/>
    <property name="binDir" location="bin"/>
    <property name="jarDir" location="${binDir}/lib"/>
    <property name="jarFile" location="${jarDir}/AntProject.jar"/>

    <target name="Initialization">
        <delete dir="${binDir}"/>
        <delete dir="${jarDir}"/>
        <mkdir dir="${binDir}"/>
        <mkdir dir="${jarDir}"/>
    </target>

    <target name="Compilation" depends="Initialization">
        <javac srcdir="${srcDir}"
            destdir="${binDir}"/>
    </target>

    <target name="Jar" depends="Initialization, Compilation">
        <jar destfile="${jarFile}" basedir="${binDir}"/>
    </target>

    <target name="Build" depends="Initialization, Compilation, Jar">
        <echo message="Ant is building your project."/>
    </target>

</project>
```

当把这个 XML 文件输入 *build.xml* 中时,在 Outline 视图中可以看到其中的属性和目标,如图 7-3 所示。

下一步是使用这个编译文件来编译应用程序,这一过程将在下一节中讨论。

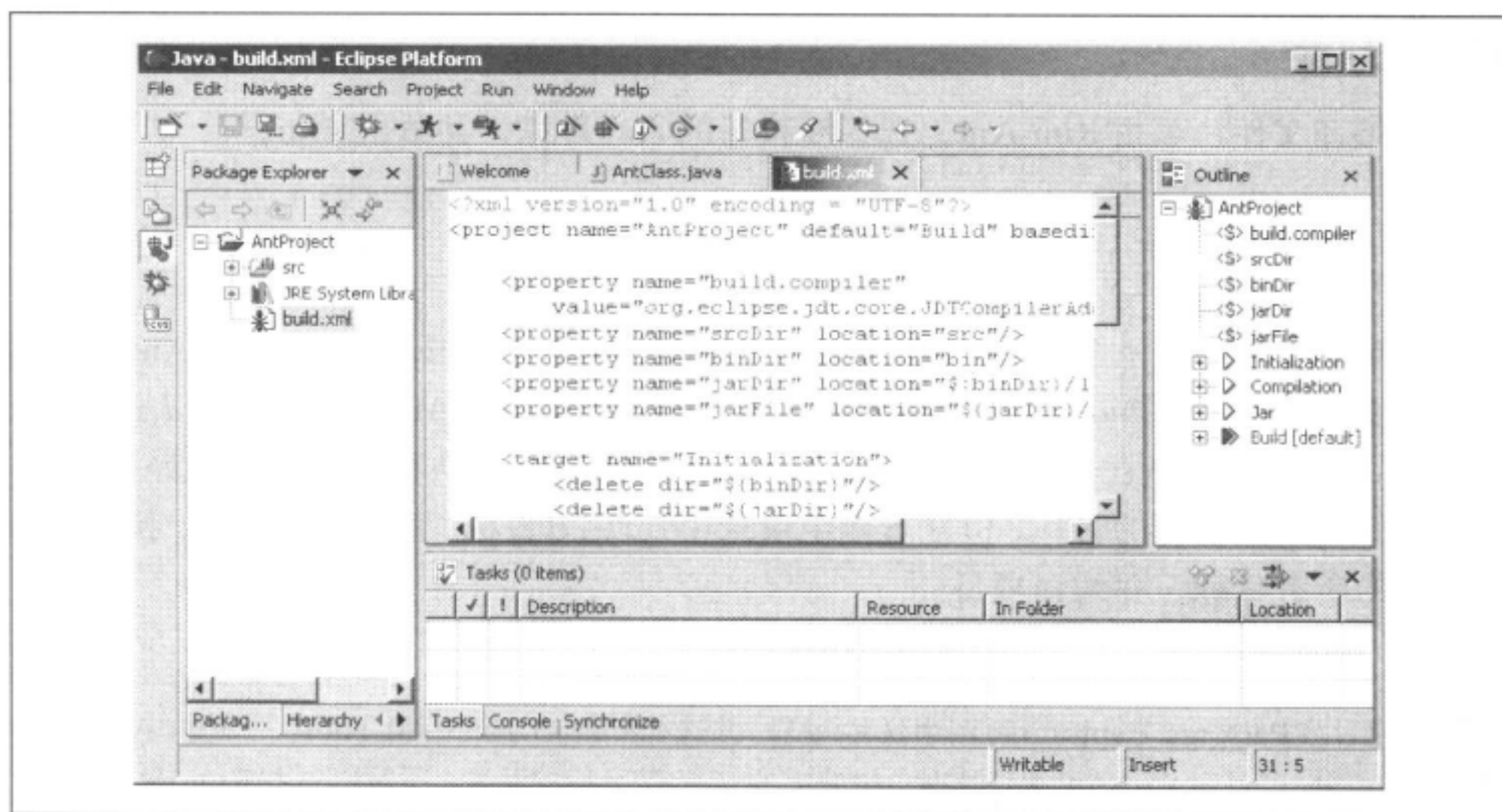


图 7-3: Eclipse 中的 build.xml 文件

注意：Eclipse 可以自动创建某些 Ant 编译文件。如果项目中使用了基于 XML 的、用于编译插件的那种清单文件（参阅第 12 章），所需要做的就是右击该文件，并单击 Create Ant Build File。

Eclipse 3.0

在 Eclipse 3.0 中，Ant 编辑器为 Ant 提供了更多的支持，包括属性、目标和引用对象（如路径）的工具提示。现在，使用 Ant 编辑器快捷菜单中的 Format 命令（Ctrl-Shift-F），还可以重新格式化 Ant 编译文件。

参考

7.2 节，使用 Ant 编译 Eclipse 应用程序；7.4 节，使用不同的编译文件；*Ant: The Definitive Guide* (O'Reilly)；*Java XP Cookbook* (O'Reilly)；*Eclipse* (O'Reilly) 一书的第 5 章。

7.2 使用 Ant 编译 Eclipse 应用程序

问题

你有一个 Ant 编译文件，并且你想编译自己的应用程序。

解决方案

右击该编译文件，单击 Run Ant，选择要执行的 Ant 目标，并单击 Run。

讨论

在上一节中，我们为一个 Eclipse 项目创建了编译文件；要在 Eclipse 自带的 Ant 版本中运行该文件，可右击 *build.xml*，并单击 Run Ant。这将打开 AntProject *build.xml* 对话框，如图 7-4 所示。从图中可以看到所创建的 Ant 目标，且默认目标 Build 已经被选中。可以选择要独立运行的目标，但在本例中仅选中 Build 目标，然后单击 Run，执行编译文件中的所有目标，并编译项目。

注意： 通过在 Package Explorer 视图中选中项目，并选择 Run → External Tools → Run As → Ant Build，也可以启动 Ant 编译文件。如果需要选择要运行的目标，还可以选择 Run → External Tools → External Tools → Targets，选中要运行的目标，并单击 Run。

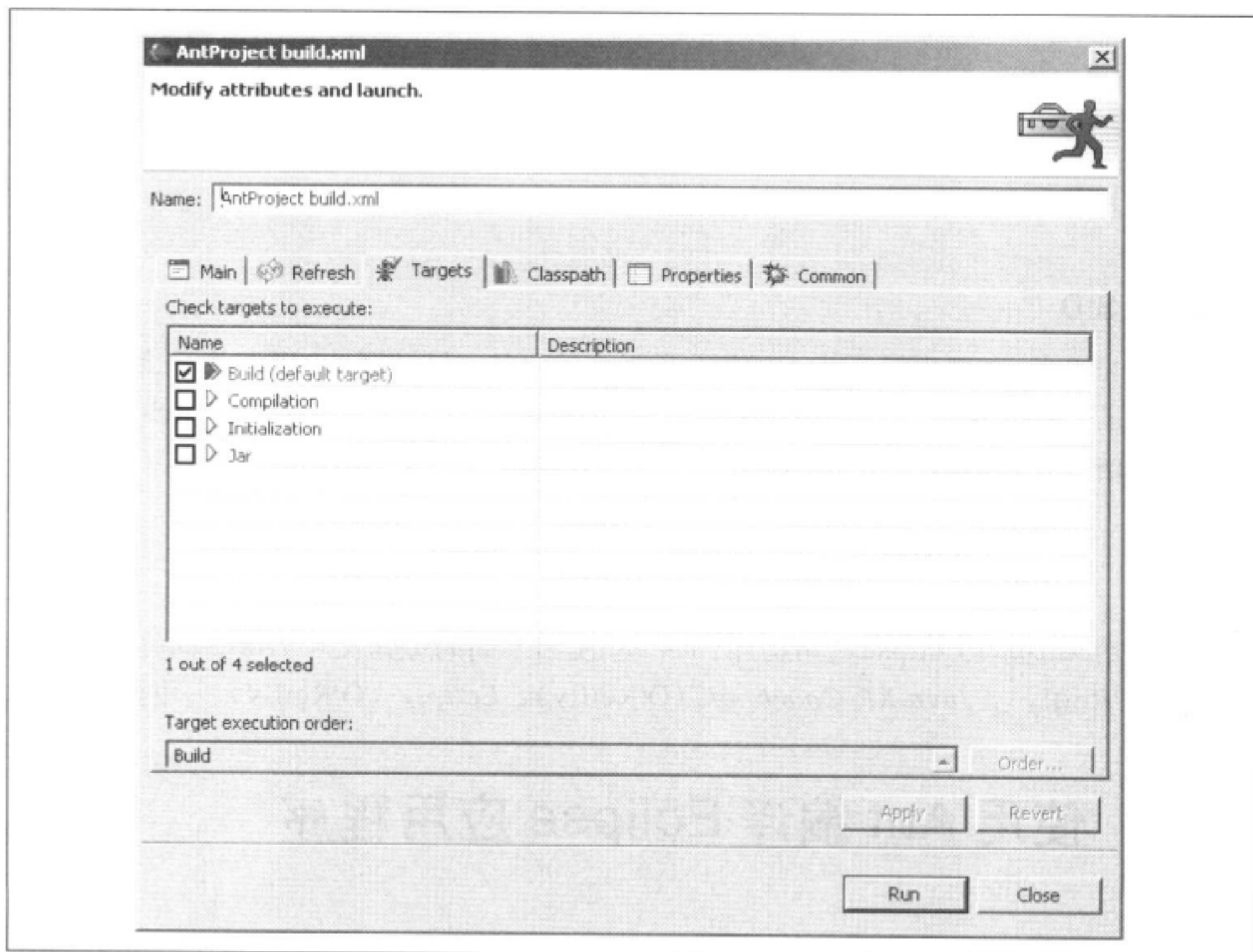


图 7-4：运行 Ant

结果显示在 Eclipse 中, 如图 7-5 所示。在这里, 注意 Ant 回送到 Console 视图中的消息 “Ant is building your project.”, 紧接其后的是消息 “BUILD SUCCESSFUL”。

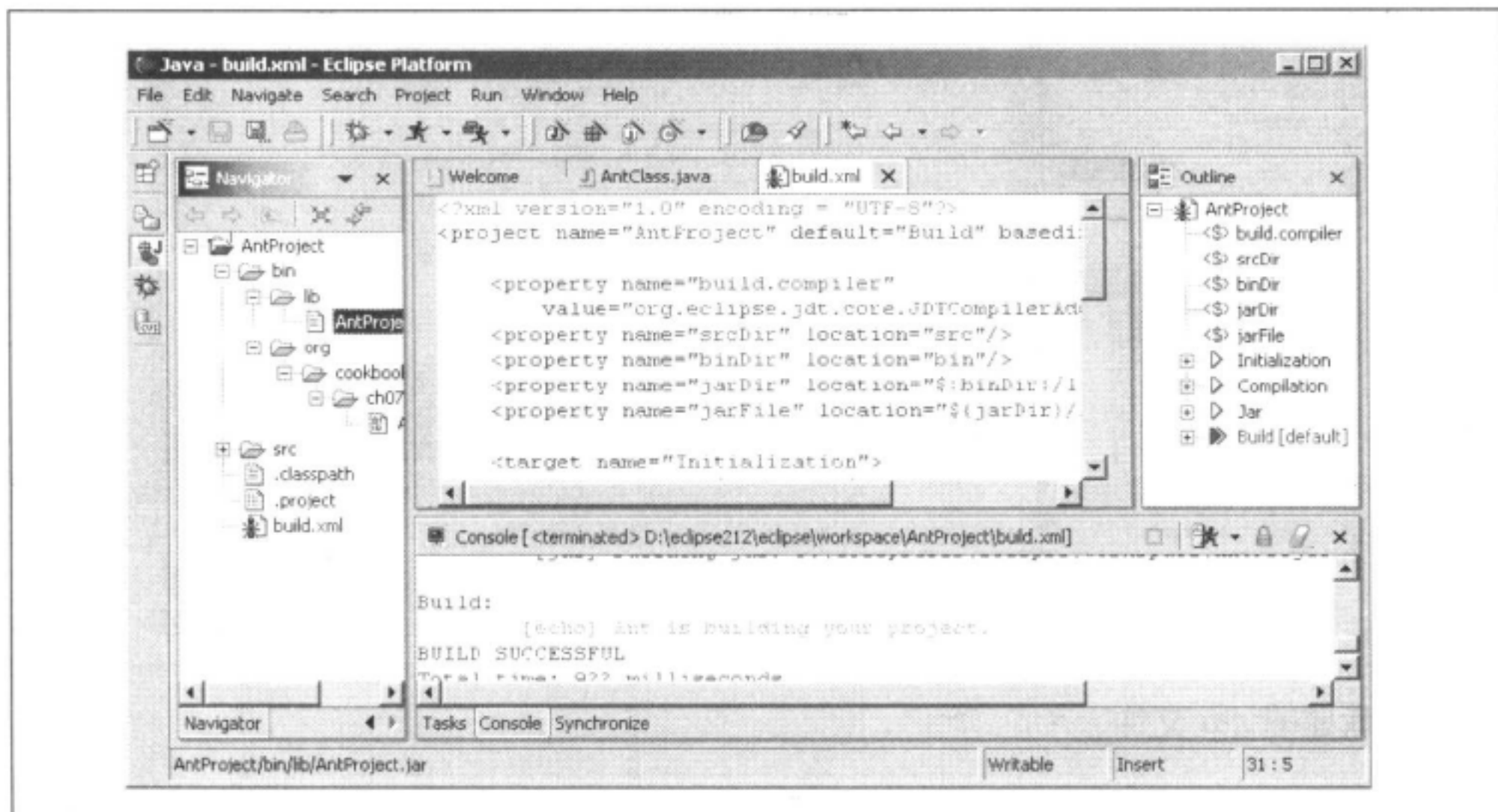


图 7-5: 一次成功的 Ant 编译

下面是 Ant 执行每个目标后在 Console 视图中显示的文本:

```
Buildfile: D:\eclipse212\eclipse\workspace\AntProject\build.xml

Initialization:
[delete] Deleting directory D:\eclipse212\eclipse\workspace\AntProject\bin
[mkdir] Created dir: D:\eclipse212\eclipse\workspace\AntProject\bin
[mkdir] Created dir: D:\eclipse212\eclipse\workspace\AntProject\bin\lib

Compilation:
[javac] Compiling 1 source file to D:\eclipse212\eclipse\workspace\AntProject\bin
[javac] Compiled 22 lines in 381 ms (57.7 lines/s)
[javac] 1 .class file generated

Jar:
[jar] Building jar: D:\eclipse212\eclipse\workspace\AntProject\bin\lib\AntProject.jar

Build:
[echo] Ant is building your project.
BUILD SUCCESSFUL
Total time: 922 milliseconds
```

如果切换到 Navigator 视图, 可以看到新建的 *lib* 目录, 其中包含 *.jar* 文件 *AntProject.jar*, 如图 7-6 所示。

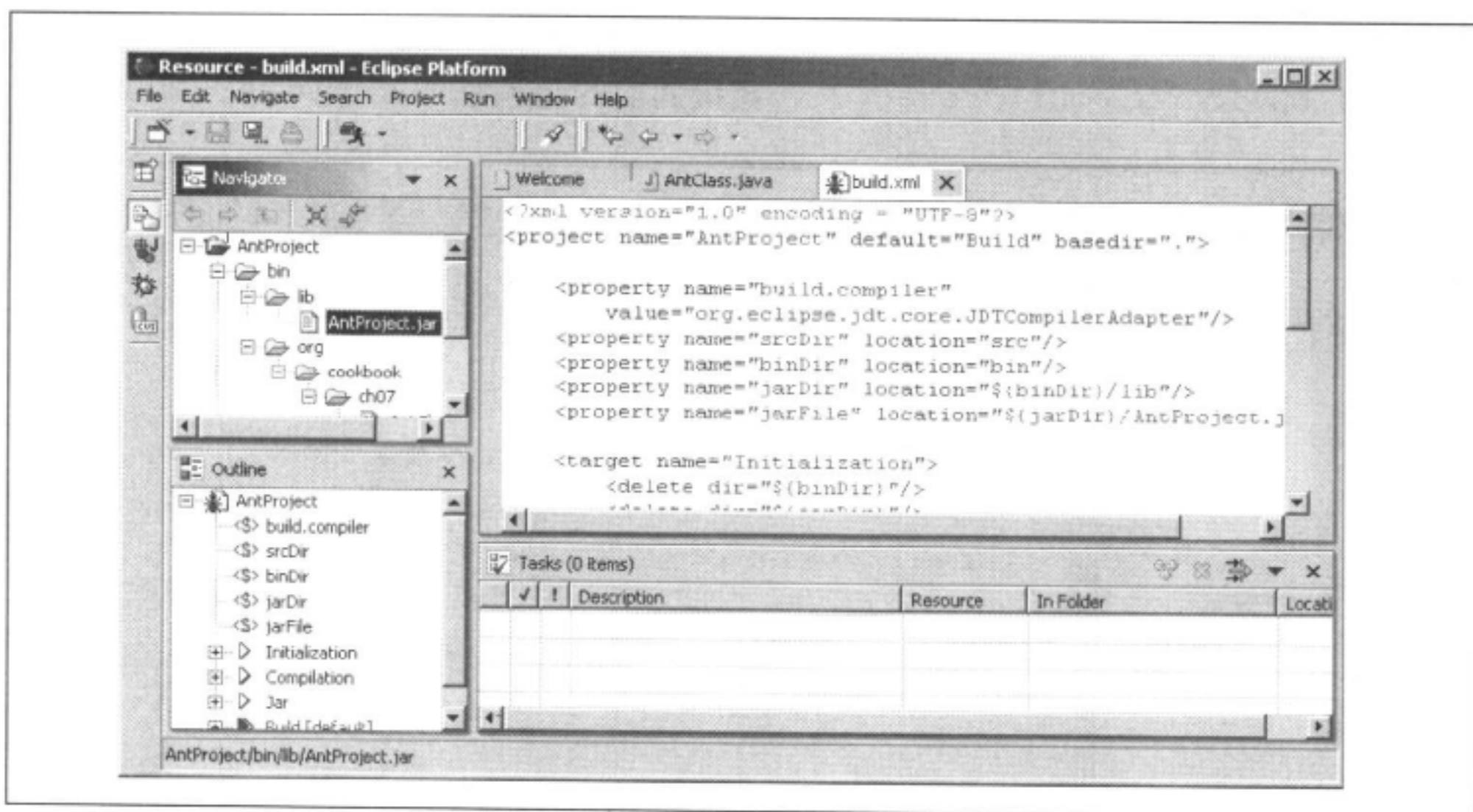


图 7-6: 新建的 .jar 文件

Eclipse 3.0

在 Eclipse 3.0 中, 默认情况下, Ant 在运行 Eclipse 的 Java 虚拟机 (JVM) 中执行。也就是说, 不必设置 `build.compiler` 属性 (本例中我们设置了该属性), 以便使用与 Eclipse 相同的 JVM。在 Eclipse 3.0 中, 通过右击 `build.xml` 文件, 单击 JRE 标签, 然后单击 Run Ant, 可以设置 Ant 使用的 JVM。

参考

Eclipse (O'Reilly) 一书的第 5 章。

7.3 捕获 Ant 编译文件语法问题

编译文件中存在一个问题, 而你想跟踪这个问题。

解决方案

Ant 的当前版本仅在你试图运行 Ant 脚本时才显示其中的语法错误。然而, Eclipse 3.0 可以在语法错误出现时捕获并显示它们。

注意：这是一个只适用于 Eclipse 3.0 的解决方案。

讨论

在编写 Ant 脚本时，Eclipse 确实可以为捕获语法错误提供一些支持。然而，只有在 XML 文档的 XML 意义不合适时（如嵌套错误、遗漏属性值的引号等），Eclipse 编辑器才捕捉语法错误。图 7-7 中有一个例子，其中遗漏了文本 `jarFile` 前面的引号，而文本 `jarFile` 是要赋给 `<property>` 元素中的一个属性的。

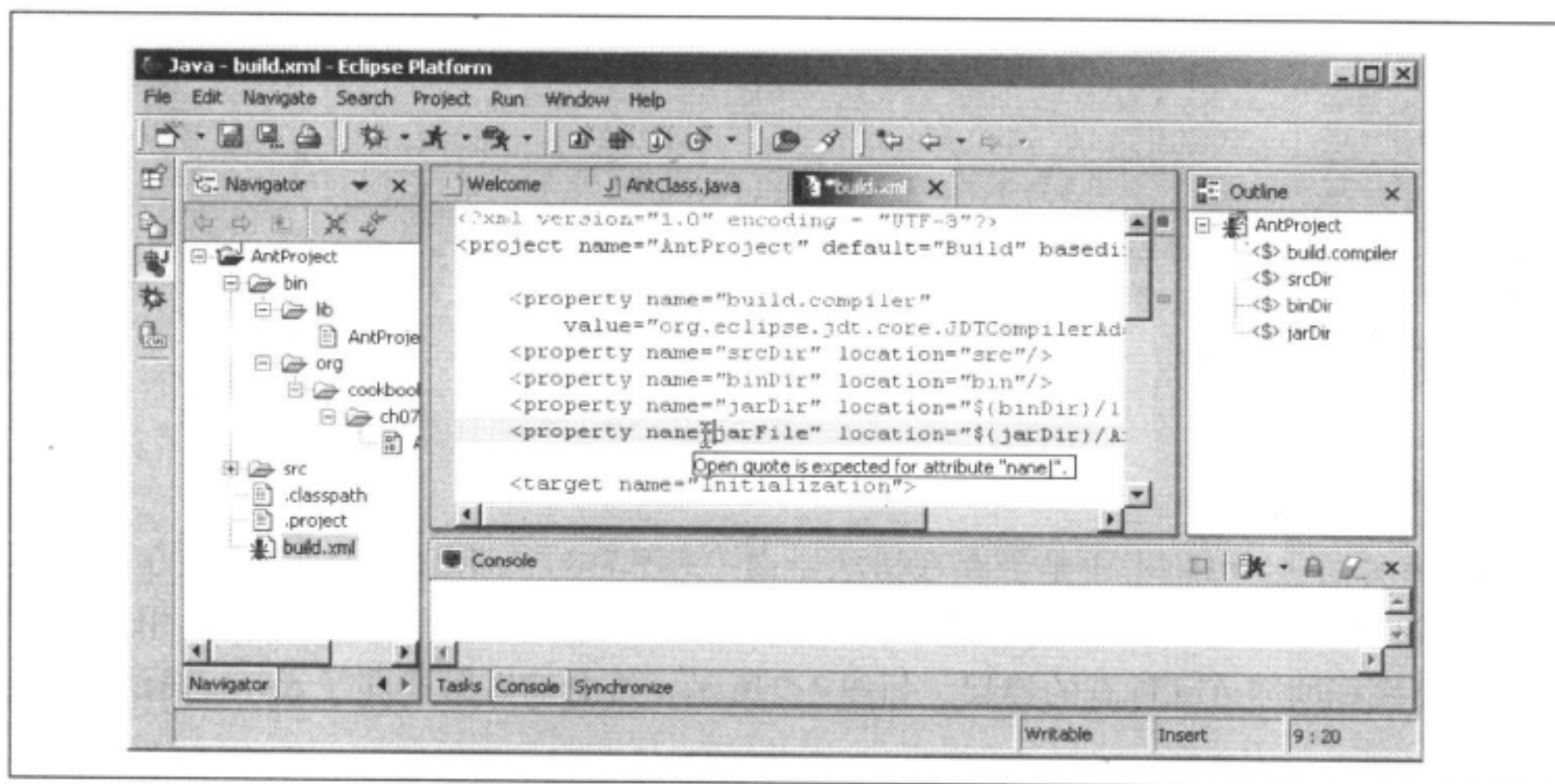


图 7-7：Ant 脚本中的一个 XML 错误

然而，需要注意的是，尽管 Ant 编辑器指出了该 XML 文档的格式错误，但却完全忽略了这一事实：`<property>` 元素的 `name` 属性被误拼为 `nane`，如图 7-7 所示。

另一方面，如果尝试运行该脚本，Ant 会发现存在一个语法错误，并得到如图 7-8 所示的结果，其中指出了该语法错误。

当你尝试运行脚本时，Eclipse 会显示该语法错误，但以后就要靠你自己了；Quick Fixes 功能不可用。你也不能以交互方式调试 Ant 脚本。

注意：实际上，值得注意的是，Ant 编辑器确实为 Ant 编译文件提供了代码助手。例如，如果输入符号 `<` 并暂停，代码助手将列出可用的 Ant 编译文件元素，可以选择一个元素。如果把光标置于一个 Ant 元素的起始标记内，并按组合键 `Ctrl-Space`，代码助手还将列出 Ant 元素的可用属性。

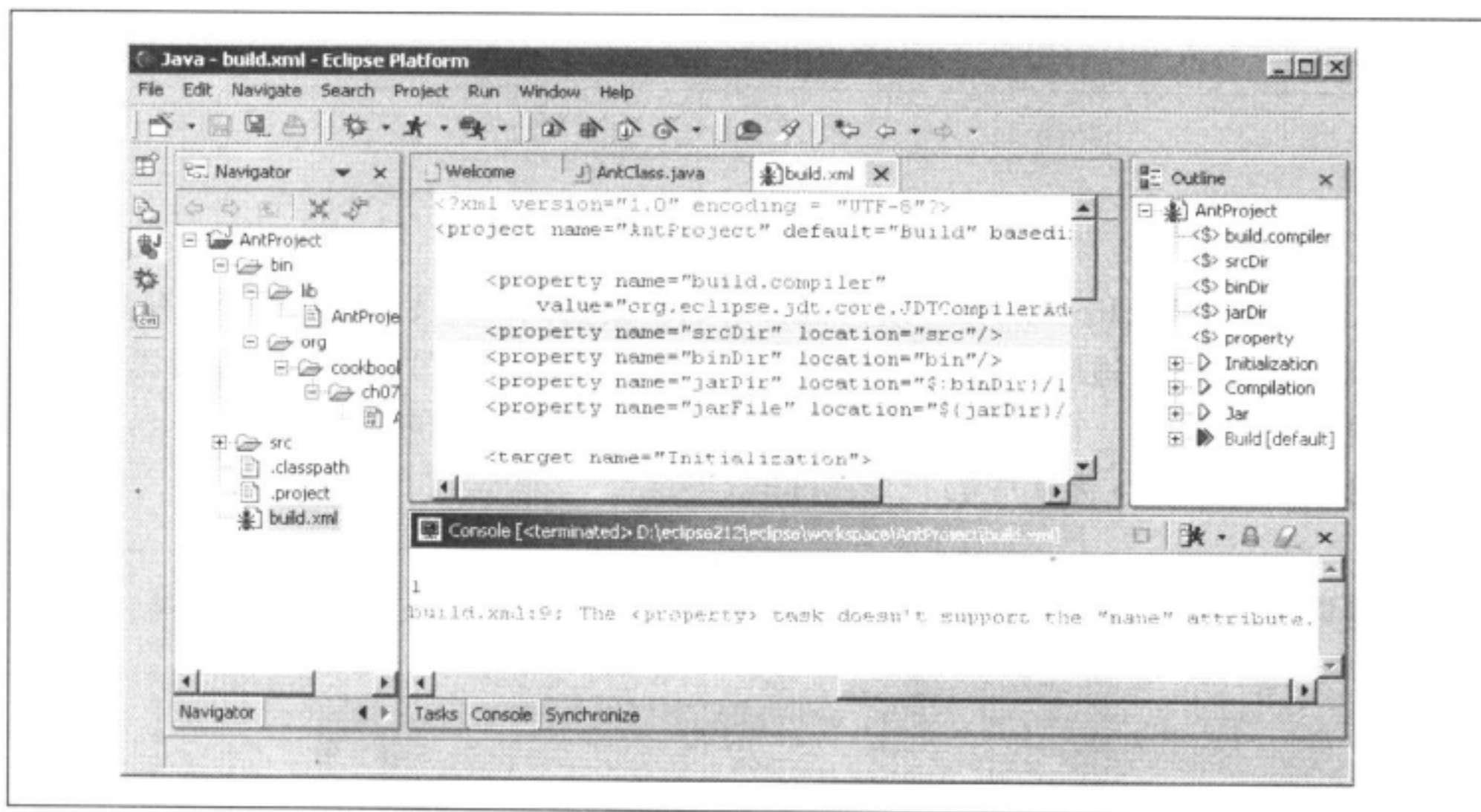


图 7-8：一个 Ant 错误

Eclipse 3.0

Eclipse 确实知道 Ant 脚本中是否存在语法错误（见图 7-8），那么它为什么不能在你编写脚本时就指出错误呢？在 Eclipse 3.0 中，这一功能实现了。现在，Ant 编辑器用典型的红色波浪线标明了 Ant 特有的错误，如图 7-9 所示，其中编辑器捕获了 `nane` 语法错误。

这是一个相当大的改进。然而，Eclipse 开发人员仍然期待着 Ant 脚本的交互式调试和能够以可视化的方式创建脚本的向导。也许有一天我们会看到这些新特性。

参考

Eclipse (O'Reilly) 一书的第 5 章。

7.4 使用不同的编译文件

问题

你想使用一个文件名不同于 *build.xml* 的 XML 编译文件编译项目。

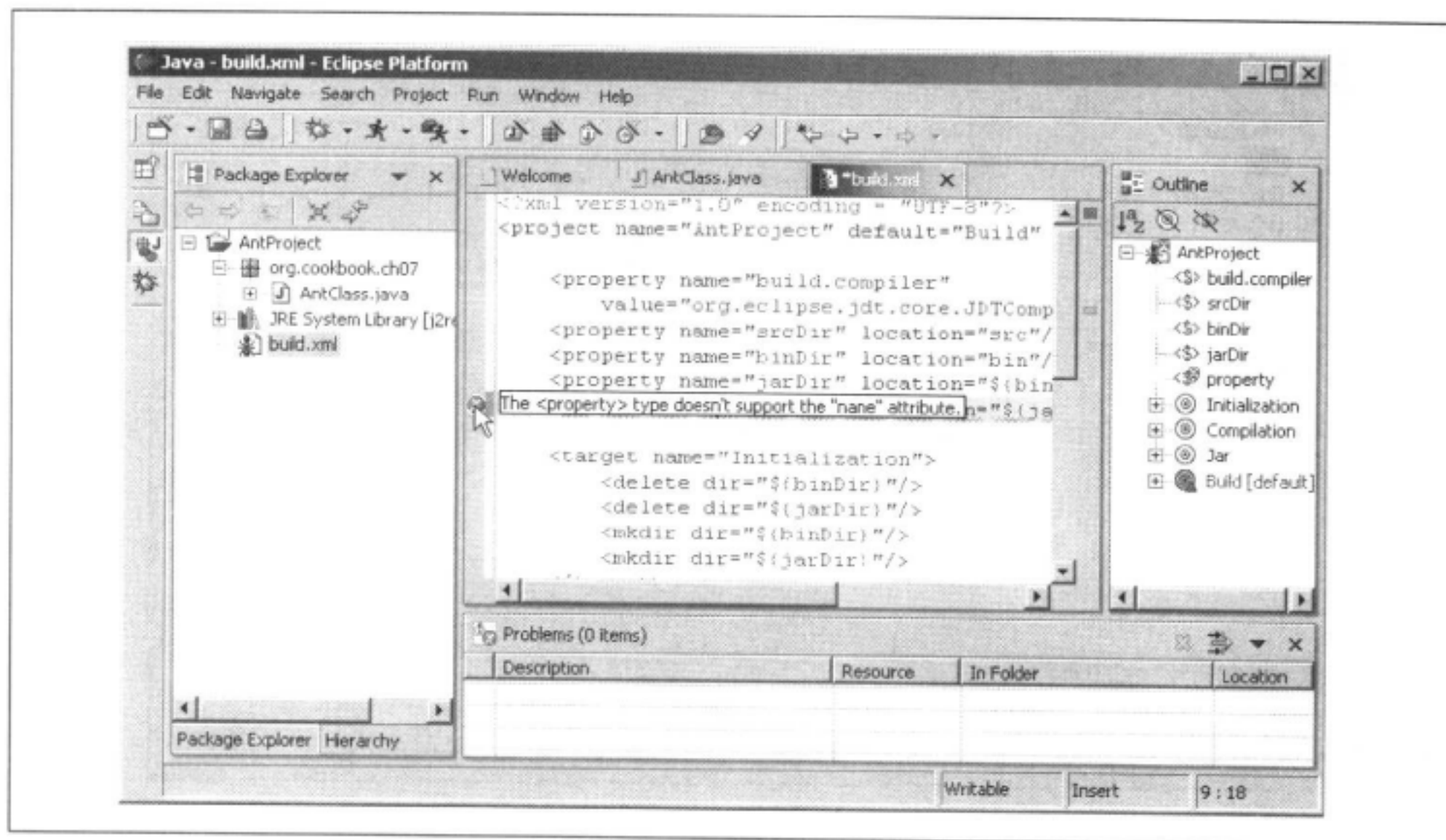


图 7-9：捕获一个语法错误

解决方案

只需选择 Window → Preferences → Ant，并输入你的编译文件名称，然后单击 OK 即可。

讨论

如果需要将编译文件的名称从 *build.xml* 改为其他的名称，可选择 Window → Preferences → Ant，打开 Preferences 窗口的 Ant 页，如图 7-10 所示。只需输入你的编译文件的名称，然后单击 OK 即可。如果需要列出多个文件名，可以用逗号隔开（每个项目只使用其中一个编译文件，否则，Eclipse 将产生混淆，引起错误）。注意，这是一个全局性的变化，如果在此修改了编译文件名，将会影响到所有项目。

注意： 还可以指定编译文件的位置。右击 *build.xml*，单击 Run Ant，并在所打开的对话框中单击 Main 标签。在此对话框中，可以设置要使用的编译文件的位置以及该编译文件的基本目录。

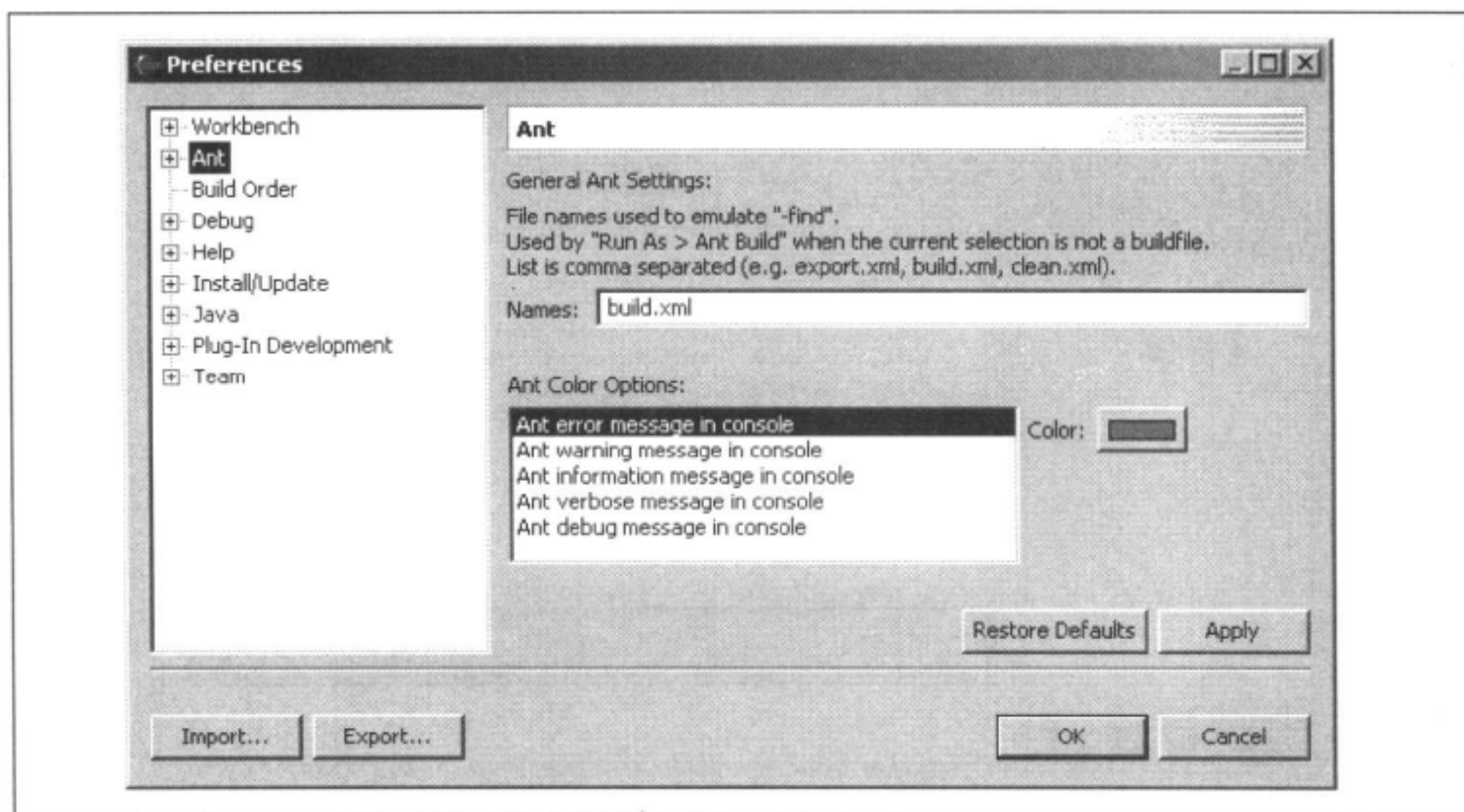


图 7-10: 设置编译文件名

注意，还可以在 Window → Preferences → Ant 对话框中对 Ant 进行些许配置。在这个对话框中只能选择在 Console 视图中显示的 Ant 消息的颜色，人们对这一功能印象不深。通过选择 Ant 项目下的节点，可以得到更多的选项；详细内容请参阅以下几节。

Eclipse 3.0

在 Eclipse 3.0 中，Window → Preferences → Ant 窗口中增加了一些选项，可以处理一些附加的警告和错误，比如当 Ant 类路径没有包含 *tools.jar* 文件时，如图 7-11 所示。

参考

7.5 节，使用你自己的 Ant 版本；*Eclipse* (O'Reilly) 一书的第 5 章。

7.5 使用你自己的 Ant 版本

问题

Ant 的一个新版本，一个比 Eclipse 自带的 Ant 更新的版本推出了，而你想进行升级。或者你只是想使用一个与 Eclipse 自带的不同的 Ant 版本。

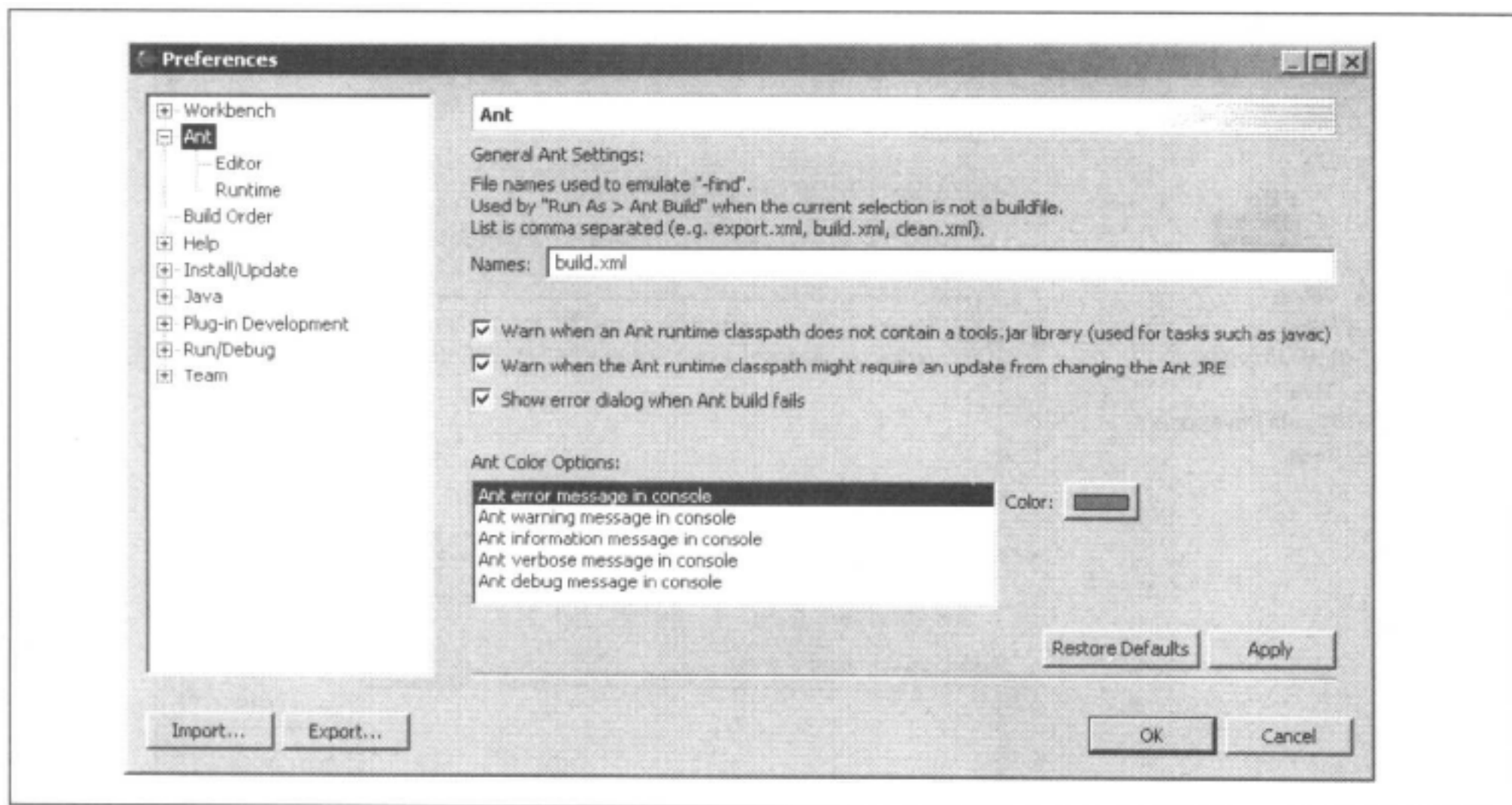


图 7-11: Eclipse 3.0 中的 Window → Preferences → Ant 窗口

解决方案

选择 Window → Preferences → Ant → Runtime → Classpath，并指向新的 Ant .jar 文件。

讨论

要设置 Ant 的版本（可以直接从 Apache 的站点 <http://ant.apache.org> 获得），可选择 Window → Preferences → Ant → Runtime → Classpath。你所需要做的只是指向新的 Ant .jar 文件 *ant.jar* 和 *optional.jar*，如图 7-12 所示。

注意： Ant 的 .jar 文件 *ant.jar* 和 *optional.jar* 是 Ant 1.5.3 特有的，这是 Eclipse 2.1.2 自带的 Ant 版本。不同的 Ant 版本可能使用不同的 .jar 文件名称，或者包含更多的 .jar 文件。

Eclipse 3.0

Eclipse 3.0 的最新版自带 Ant 1.6.0，而且具有 29 个 Ant .jar 文件（不仅仅是早期版本的两个）的类路径条目。另外，默认情况下，如果试图在 Ant 类路径上没有包含 *tools.jar* 文件的情况下执行 Ant，或者当 Ant 被配置为在与 Eclipse 相同的 JVM 上运行时（如果你的 Eclipse 面向另一个版本的 Java，你可能不想使用相同的 JVM），或者 Ant 类路径上没有包含 Xerces 包（已不再需要）时，都将收到警告消息。

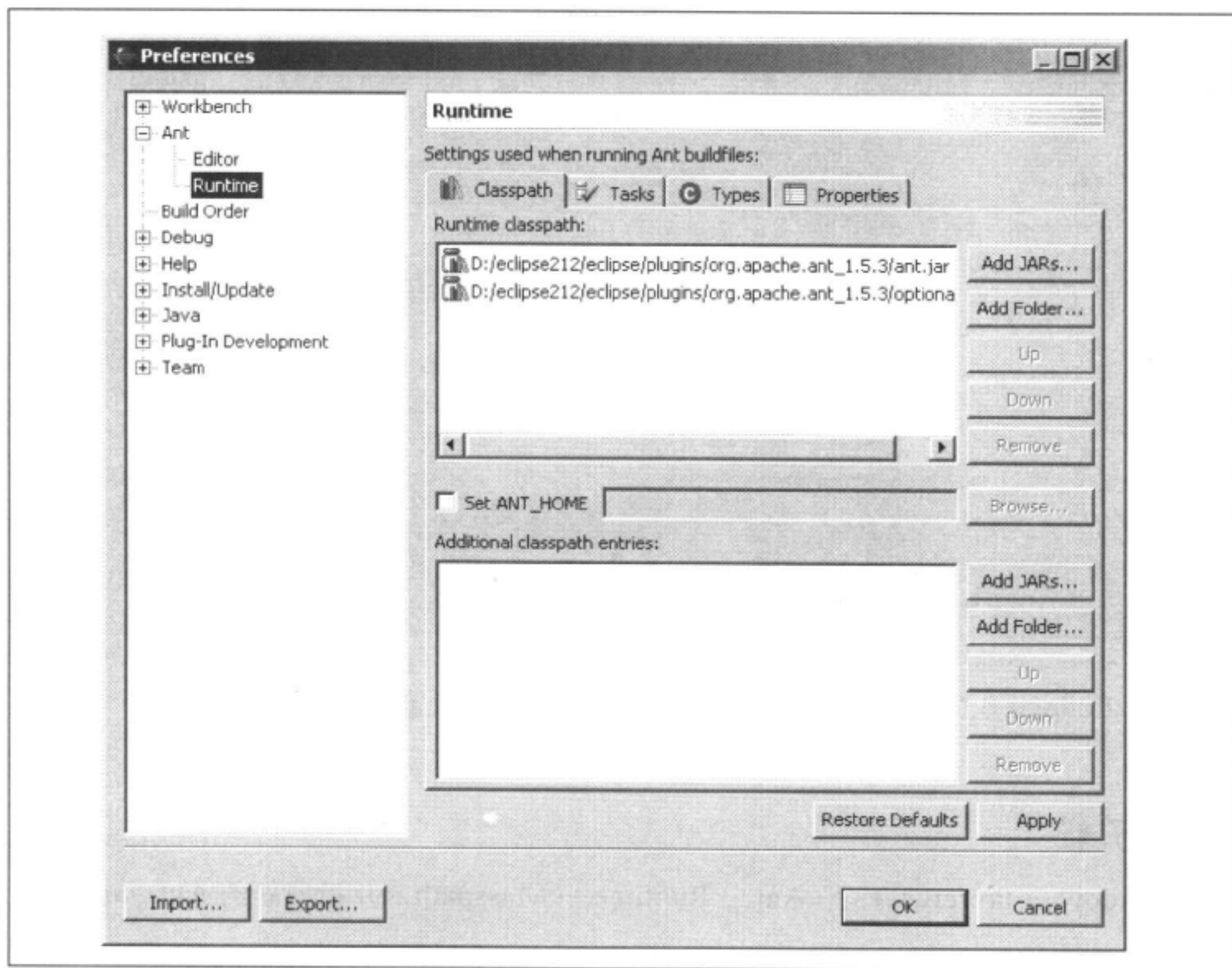


图 7-12：使用自己的 Ant 版本

参考

7.10 节，将 Ant 作为外部工具使用。

7.6 设置类型和全局属性

问题

你想为一个 Ant 编译文件创建任务和类型，或者设置 Ant 的全局属性。

解决方案

选择 Window → Preferences → Ant → Runtime，并单击 Tasks、Types 或 Properties 标签。

讨论

如果单击 Tasks 和 Types 标签，则可以添加新的 Ant 任务和类型。然后，不必使用 `<taskdef>` 或 `<typedef>` 元素，就可以在编译文件中使用这些任务和类型。在此对话框中，如果单击 Properties 标签，还可以设置 Ant 的全局属性，这些全局属性适用于所有的项目。要添加新的全局属性，只需单击 Properties 标签页上的 Add 按钮，并输入新属性的名称和值即可。

7.7 设置 Ant 编辑器的选项

问题

你想自定义 Ant 编辑器的外观。

解决方案

选择 Window → Preferences → Ant → Editor，并从该对话框的选项中进行选择。

讨论

使用 Window → Preferences → Ant → Editor 对话框，可以对 Ant 编辑器的外观进行一些控制。在 Eclipse 2.1.2 中，Ant 编辑器其实就是一个简单的 XML 编辑器，但可以指定语法加亮显示时所采用的颜色以及编辑器是否显示总标尺或行号等，如图 7-13 所示。

例如，打开行号选项可以使 Ant 编辑器显示行号，如图 7-14 所示。

7.8 设置 Ant 参数

问题

你需要传递一些参数给 Ant。

解决方案

右击编译文件，单击 Run Ant，单击 Main 标签，并输入需要使用的 Ant 参数即可。

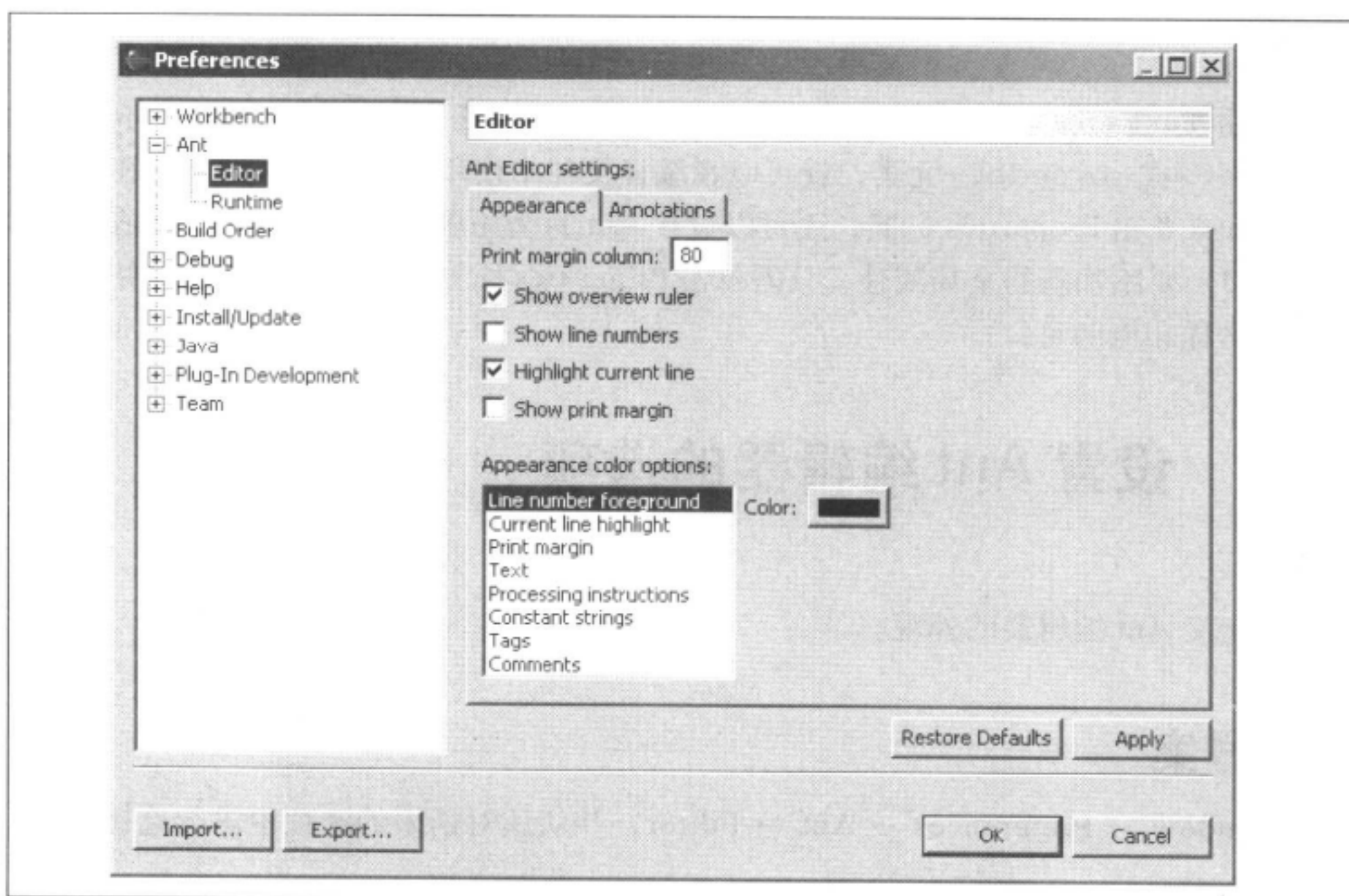


图 7-13: 设置 Ant 编辑器的选项

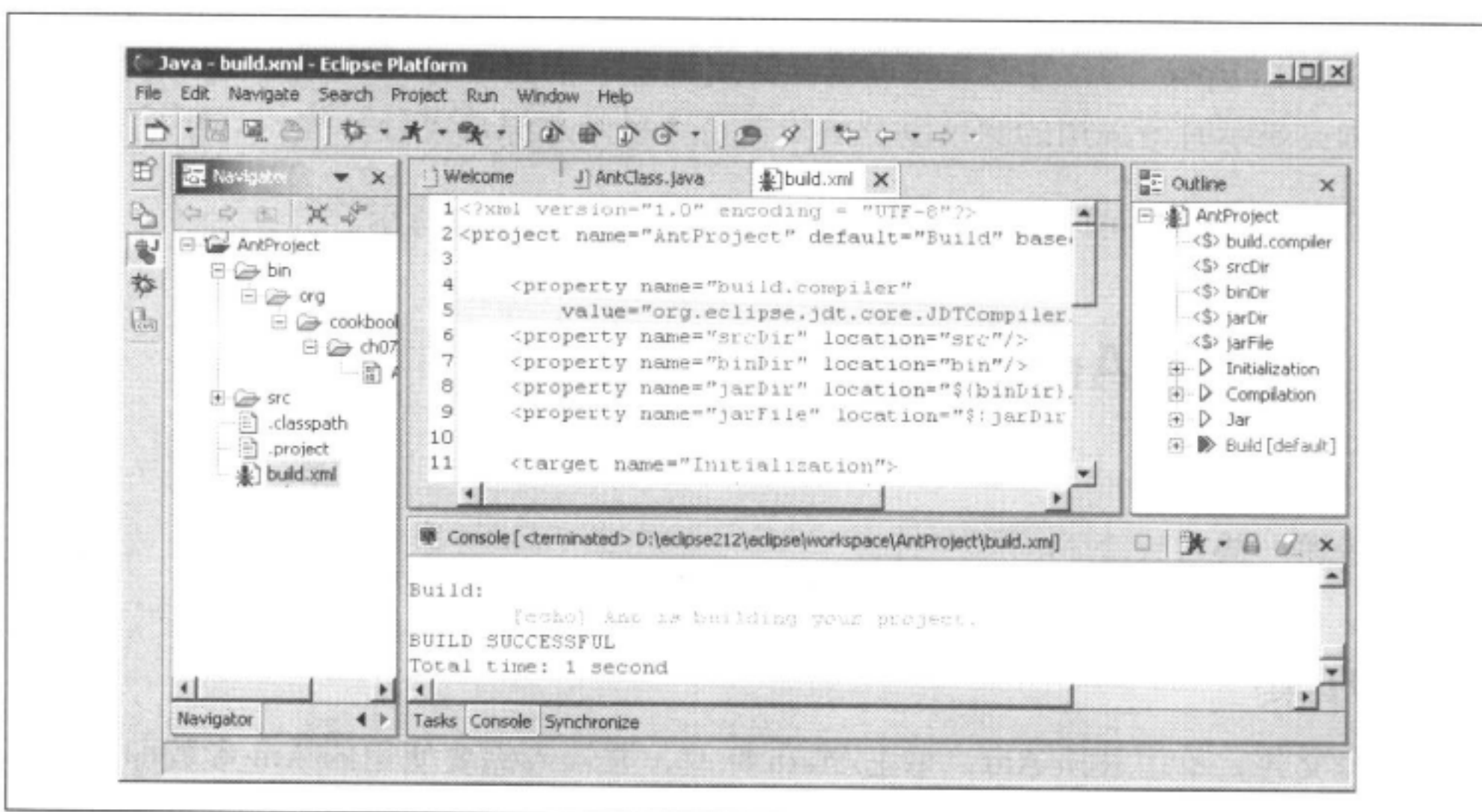


图 7-14: 在 Ant 编辑器中显示行号

讨论

如果需要传递参数给 Ant，可以右击编译文件，单击 Run Ant，并单击 Main 标签，然后在打开的对话框中的 Arguments 文本框中输入参数。这个对话框如图 7-15 所示（注意，在这个对话框中还可以设置编译文件的位置和基本目录）。

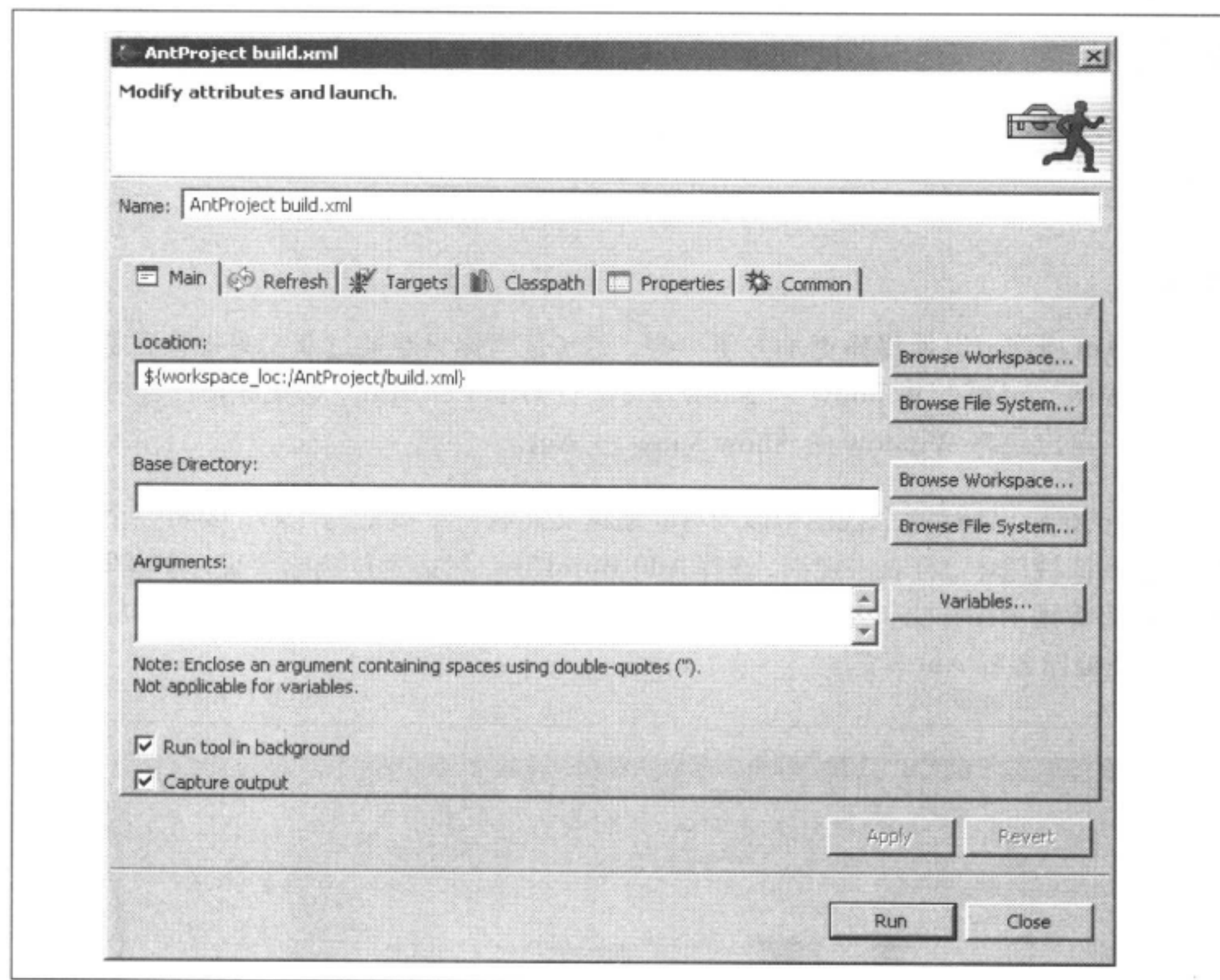


图 7-15：AntProject build.xml 对话框

Eclipse 3.0

在 Eclipse 3.0 中，如图 7-15 所示的对话框中增加了一个标签页，即 JRE 标签页，通过这个标签页可以选择 Ant 使用的 JVM。

7.9 使用 Ant 视图

问题

你想用一种更容易的方式编译 Ant 目标。

解决方案

选择 Window → Show View → Other → Ant, 打开 Ant 视图。当需要编译一个目标时, 只需在 Ant 视图中双击该目标, 或者右击它并单击 Run 即可。

讨论

较新的 Ant 视图可以使操作更加方便一些, 尤其是当你只想通过双击来编译目标时。要打开该视图, 可选择 Window → Show View → Other → Ant (如果此前已经打开了这个视图, 可以选择 Window → Show View → Ant)。

如图 7-16 所示, 通过这个视图可以对 Ant 编译文件有一个概括的了解。要将一个编译文件添加到该视图中, 可右击视图, 选择 Add Buildfile, 然后定位到需要显示的编译文件。这个视图将显示编译文件中的目标; 可以通过双击这些目标, 或者右击这些目标并选择 Run, 来编译各种 Ant 目标。

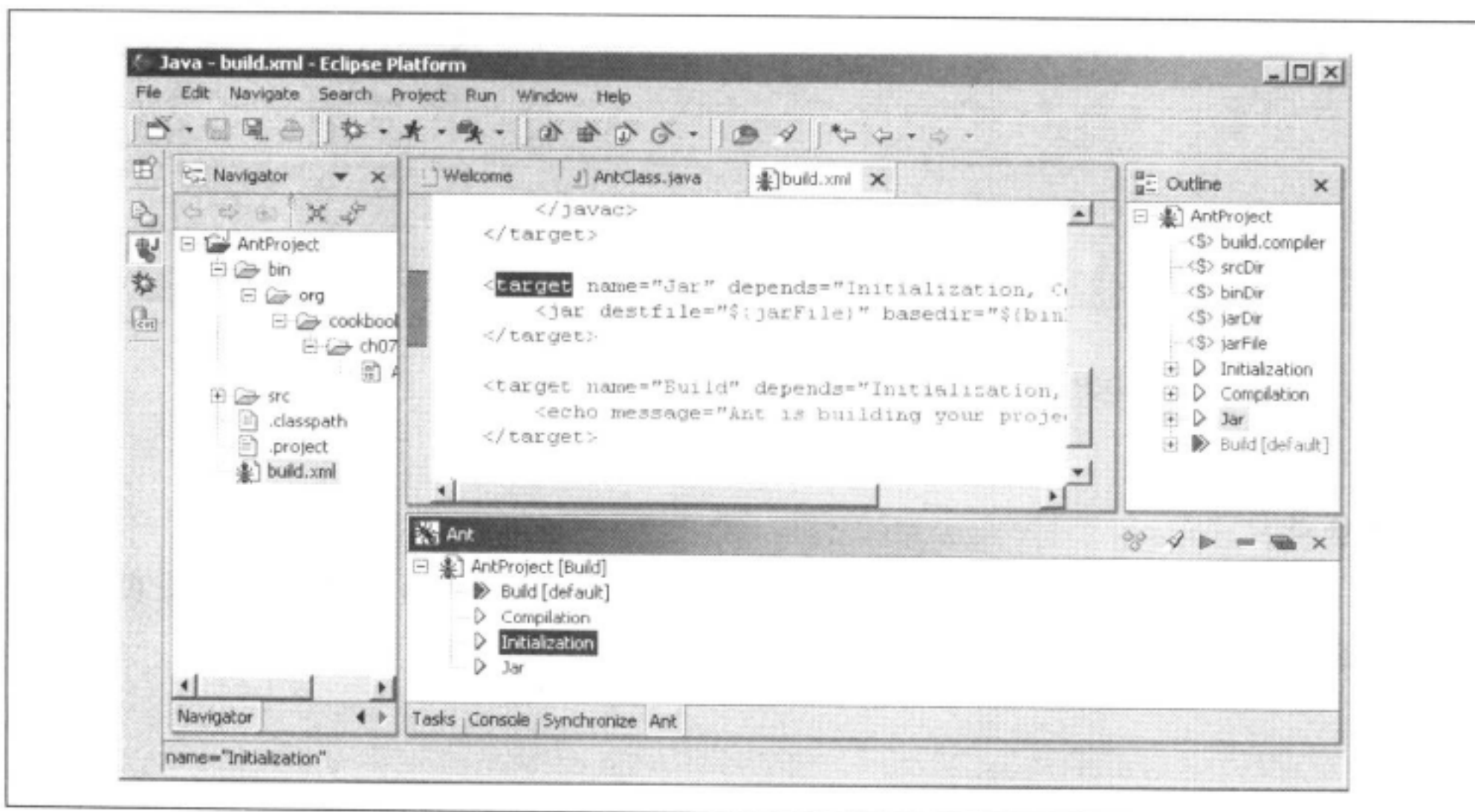


图 7-16: Ant 视图

Eclipse 3.0

在 Eclipse 3.0 中，还可以通过拖放的方式将编译文件添加到 Ant 视图中。另外，Ant 视图现在增加了一个切换按钮，通过该按钮可以过滤掉内部的编译目标。

7.10 将 Ant 用作外部工具 问题

你想把 Ant 作为外部工具来运行。

解决方案

选择 Run → External Tools → External Tools，单击 Program，单击 New，并输入将 Ant 作为外部工具安装所需的信息。然后单击 Run，执行编译文件。

讨论

Eclipse 可以启动并运行外部工具，与使用内部工具一样简单。如果选择 Run → External Tools → External Tools，单击 Program，单击 New，并输入用于代表外部 Ant 版本的名称，即可把 Ant 作为外部工具来运行。至于 Location 字段，单击 Browse File System，并找到操作系统运行 Ant 所需的正确文件（例如，在 Windows 中，是 Ant 的 *bin* 文件夹中的 *ant.bat* 文件）。在 Working Directory 字段中，输入编译文件的目录，并在 Arguments 目录中输入要传递给 Ant 的参数。然后单击 Run，执行编译文件，并进行编译。

注意：现在，外部工具可以由一个独立的线程在后台运行。只需在 Run → External Tools → External Tools 对话框中选中 Run tool in background 复选框即可。这样设置之后，可以运行 Ant 编译文件或运行一个外部程序，同时可以继续使用 Eclipse 进行工作。

参考

7.5 节，使用你自己的 Ant 版本。

第 8 章

SWT：文本、按钮、列表和非矩形窗口

8.0 简介

IBM 开发了标准窗口小部件工具箱（Standard Widget Toolkit, SWT），将其作为抽象窗口工具箱（Abstract Windowing Toolkit, AWT）和 Java 中的 Swing 组件集的替代工具。SWT 是一个功能全面的 GUI API，而且内置到了 Eclipse 中。在本章以及接下来的两章中，将重点介绍 SWT，并提供一些很有特色的代码。

Java 长期以来一直支持 GUI。早期的工作之一是 Java 小程序，例 8-1 中的 Eclipse 项目 *Applet* 是一个示例小程序。

例 8-1：一个示例小程序

```
package org.cookbook.ch08;

import java.applet.Applet;
import java.awt.*;

public class AppletClass extends Applet {

    public void init()
    {
        setBackground(Color.white);
    }

    public void start()
    {
    }

    public void paint(Graphics g)
    {
        g.drawString("Greetings from Applets.", 40, 100);
    }
}
```

```
public void stop()
{
}

public void destroy()
{
}
}
```

选择 Run → Run As → Java Applet，可以从 Eclipse 中运行这个小程序。其运行结果如图 8-1 所示。

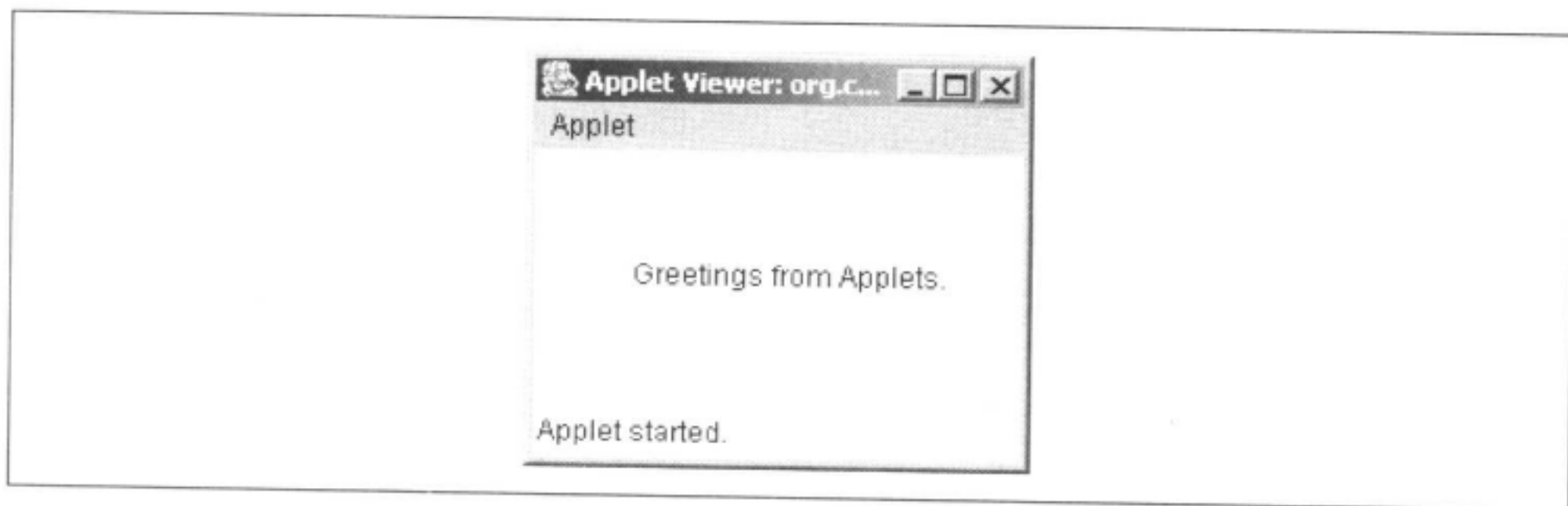


图 8-1：从 Eclipse 中运行 Java 小程序

作为一个早期的 GUI 支持，Sun 公司用数周的时间编写了 AWT，并在当时取得了相当大的成功。AWT 易于使用，并使 Java 开发人员能够创建基本的 GUI。

AWT 使用其基础操作系统的控件。例如，在 Windows 中，可以看到一个 Windows 文本框；在 Macintosh 中，可以看到一个 Mac 文本框。事实上，由于各种操作系统的控件集存在差异，Sun 只实现了所有平台通用的一组控件。而 AWT 由于对开发人员的需要的满足十分有限而寿终正寝。

为了满足开发人员日益增长的需要，Sun 创建了 Swing，一种对专用控件（如树和表）的非本机实现提供支持的 GUI API。例 8-2 有一个 Swing 示例，即名为 *SwingApp* 的 Eclipse 项目，从本书的 O'Reilly 站点的 Examples 页，可以找到这个示例项目。

例 8-2：一个 Swing 应用程序

```
package org.cookbook.ch08;

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class SwingClass extends JFrame {

    Panel panel;
```

```
public SwingClass()
{
    super("Swing Example");

    Container contentPane = getContentPane();
    panel = new Panel();
    contentPane.add(panel);
}

public static void main(String args[])
{
    final JFrame frame = new SwingClass();

    frame.setDefaultCloseOperation(DISPOSE_ON_CLOSE);
    frame.setBounds(100, 100, 280, 250);
    frame.setVisible(true);

    frame.addWindowListener(new WindowAdapter() {
        public void windowClosing(WindowEvent e) {
            System.exit(0);
        }
    });
}

class Panel extends JPanel
{
    Panel()
    {
        setBackground(Color.white);
    }

    public void paintComponent (Graphics g)
    {
        super.paintComponent(g);
        g.drawString("Greetings from Swing", 70, 100);
    }
}
```

从 Eclipse 启动一个 Swing 应用程序与启动已经运行过的应用程序没什么不同，只需选择 Run → Run As → Java Application 即可。运行结果如图 8-2 所示。

Swing 提供了许多功能，但都是 Java 特定的，至少直观上如此。Sun 曾尝试添加一个操作系统特定的“观感” (look-and-feel) 选项，但最后，Sun 未能跟上操作系统的快速变化。另外，与本机实现相比，Swing 的响应速度太慢。

SWT 在许多方面弥补了 Swing 的不足。它支持一组窗口小部件，当本机控件可用时，这些部件使用本机控件。如果本机控件不可用，SWT 就使用它自己的窗口小部件。SWT 大大提升了 IBM 在开发人员心目中的地位；它是为完全取代 AWT 和 Swing 而设计的。不出所料，SWT 提供了一组丰富而强大的特性。

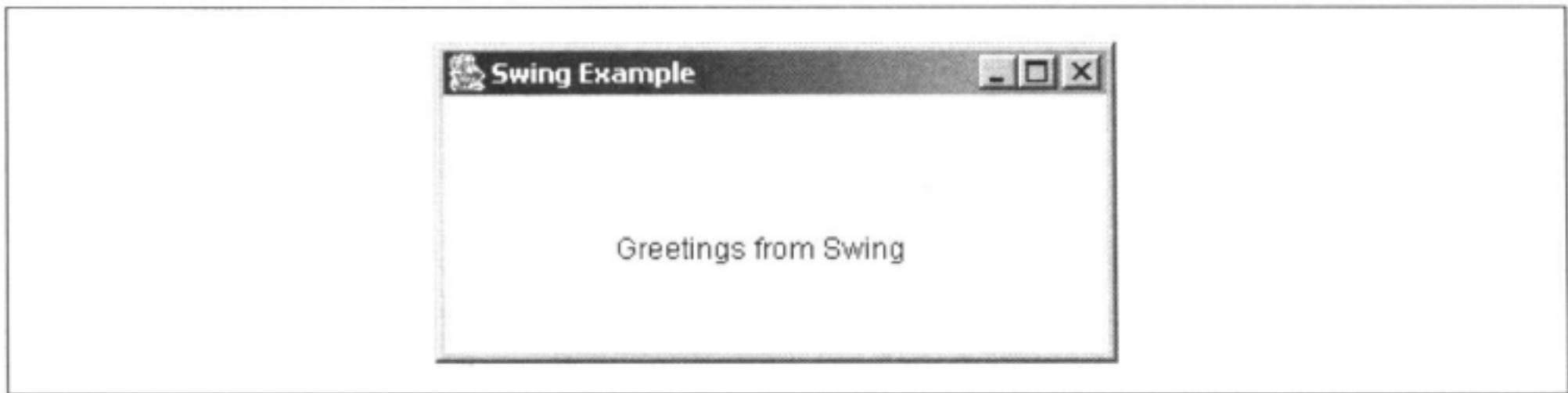


图 8-2：从 Eclipse 启动一个 Swing 应用程序

8.1 使用 SWT 窗口小部件

问题

你想使用 SWT，那你就需要知道 SWT 有什么窗口小部件是可以使用的。

解决方案

参见表 8-1，其中列出了 SWT 的所有窗口小部件及其样式和事件。

讨论

作为使用 SWT 的起点，表 8-1 提供了 SWT 的所有窗口小部件参考资料。

表 8-1：SWT 窗口小部件

窗口小部件	样式	事件
按钮	BORDER, CHECK, PUSH, RADIO, TOGGLE, FLAT, LEFT, RIGHT, CENTER, ARROW (with UP, DOWN)	Dispose, FocusIn, FocusOut, Help, KeyDown, KeyUp, MouseDoubleClick, MouseDown, MouseEnter, MouseExit, MouseHover, MouseUp, MouseMove, Move, Paint, Resize, Selection
画布	BORDER, H_SCROLL, V_SCROLL, NO_BACKGROUND, NO_FOCUS, NO_MERGE_PAINTS, NO_REDRAW_RESIZE, NO_RADIO_GROUP	Dispose, FocusIn, FocusOut, Help, KeyDown, KeyUp, MouseDoubleClick, MouseDown, MouseEnter, MouseExit, MouseHover, MouseUp, MouseMove, Move, Paint, Resize

表 8-1: SWT 窗口小部件 (续)

窗口小部件	样式	事件
插入记号		Dispose
组合框	BORDER, DROP_DOWN, READ_ONLY, SIMPLE	Dispose, FocusIn, FocusOut, Help, KeyDown, KeyUp, MouseDoubleClick, MouseDown, MouseEnter, MouseExit, MouseHover, MouseUp, MouseMove, Move, Paint, Resize, DefaultSelection, Modify, Selection
复合框	BORDER, H_SCROLL, V_SCROLL	Dispose, FocusIn, FocusOut, Help, KeyDown, KeyUp, MouseDoubleClick, MouseDown, MouseEnter, MouseExit, MouseHover, MouseUp, MouseMove, Move, Paint, Resize
控件栏	BORDER	Dispose, FocusIn, FocusOut, Help, KeyDown, KeyUp, MouseDoubleClick, MouseDown, MouseEnter, MouseExit, MouseHover, MouseUp, MouseMove, Move, Paint, Resize
控件项	DROP_DOWN	Dispose
组	BORDER, SHADOW_ETCHED_IN, SHADOW_ETCHED_OUT, SHADOW_IN, SHADOW_OUT, SHADOW_NONE	Dispose, FocusIn, FocusOut, Help, KeyDown, KeyUp, MouseDoubleClick, MouseDown, MouseEnter, MouseExit, MouseHover, MouseUp, MouseMove, Move, Paint, Resize
标签	BORDER, CENTER, LEFT, RIGHT, WRAP, SEPARATOR (with HORIZONTAL, SHADOW_IN, SHADOW_OUT, SHADOW_NONE, VERTICAL)	Dispose, FocusIn, FocusOut, Help, KeyDown, KeyUp, MouseDoubleClick, MouseDown, MouseEnter, MouseExit, MouseHover, MouseUp, MouseMove, Move, Paint, Resize
列表	BORDER, H_SCROLL, V_SCROLL, SINGLE, MULTI	Dispose, FocusIn, FocusOut, Help, KeyDown, KeyUp, MouseDoubleClick, MouseDown, MouseEnter, MouseExit, MouseHover, MouseUp, MouseMove, Move, Paint, Resize, Selection, DefaultSelection

表 8-1: SWT 窗口小部件 (续)

窗口小部件	样式	事件
菜单	BAR, DROP_DOWN, NO_RADIO_GROUP, POP_UP	Dispose, Help, Hide, Show
菜单项	CHECK, CASCADE, PUSH, RADIO, SEPARATOR	Dispose, Arm, Help, Selection
进度栏	BORDER, INDETERMINATE, SMOOTH, HORIZONTAL, VERTICAL	Dispose, FocusIn, FocusOut, Help, KeyDown, KeyUp, MouseDoubleClick, MouseDown, MouseEnter, MouseExit, MouseHover, MouseUp, MouseMove, Move, Paint, Resize
框	BORDER, HORIZONTAL, VERTICAL	Dispose, FocusIn, FocusOut, Help, KeyDown, KeyUp, MouseDoubleClick, MouseDown, MouseEnter, MouseExit, MouseHover, MouseUp, MouseMove, Move, Paint, Resize, Selection
标尺	BORDER, HORIZONTAL, VERTICAL	Dispose, FocusIn, FocusOut, Help, KeyDown, KeyUp, MouseDoubleClick, MouseDown, MouseEnter, MouseExit, MouseHover, MouseUp, MouseMove, Move, Paint, Resize, Selection
滚动条	HORIZONTAL, VERTICAL	Dispose, Selection
Shell	BORDER, H_SCROLL, V_SCROLL, CLOSE, MIN, MAX, NO_TRIM, RESIZE, TITLE (见 SHELL_TRIM, DIALOG_TRIM)	Dispose, FocusIn, FocusOut, Help, KeyDown, KeyUp, MouseDoubleClick, MouseDown, MouseEnter, MouseExit, MouseHover, MouseUp, MouseMove, Move, Paint, Resize, Activate, Close, Deactivate, Deiconify, Iconify
滑块	BORDER, HORIZONTAL, VERTICAL	Dispose, FocusIn, FocusOut, Help, KeyDown, KeyUp, MouseDoubleClick, MouseDown, MouseEnter, MouseExit, MouseHover, MouseUp, MouseMove, Move, Paint, Resize, Selection
标签文件夹	BORDER	Dispose, FocusIn, FocusOut, Help, KeyDown, KeyUp, MouseDoubleClick, MouseDown, MouseEnter, MouseExit, MouseHover, MouseUp, MouseMove, Move, Paint, Resize, Selection
制表项		Dispose

表 8-1: SWT 窗口小部件 (续)

窗口小部件	样式	事件
表格	BORDER, H_SCROLL, V_SCROLL, SINGLE, MULTI, CHECK, FULL_SELECTION, IDE_SELECTION	Dispose, FocusIn, FocusOut, Help, KeyDown, KeyUp, MouseDoubleClick, MouseDown, MouseEnter, MouseExit, MouseHover, MouseUp, MouseMove, Move, Paint, Resize, Selection, DefaultSelection
表格列	LEFT, RIGHT, CENTER	Dispose, Move, Resize, Selection
表项		Dispose
文本	BORDER, SINGLE, READ_ONLY, LEFT, CENTER, RIGHT, WRAP, MULTI (with H_SCROLL, V_SCROLL)	Dispose, FocusIn, FocusOut, Help, KeyDown, KeyUp, MouseDoubleClick, MouseDown, MouseEnter, MouseExit, MouseHover, MouseUp, MouseMove, Move, Paint, Resize, DefaultSelection, Modify, Verify
工具栏	BORDER, FLAT, WRAP, RIGHT, SHADOW_OUT, HORIZONTAL, VERTICAL	Dispose, FocusIn, FocusOut, Help, KeyDown, KeyUp, MouseDoubleClick, MouseDown, MouseEnter, MouseExit, MouseHover, MouseUp, MouseMove, Move, Paint, Resize,
工具项	PUSH, CHECK, RADIO, SEPARATOR, DROP_DOWN	Dispose, Selection
跟踪器	LEFT, RIGHT, UP, DOWN, RESIZE	Dispose, Move, Resize
树	BORDER, H_SCROLL, V_SCROLL, SINGLE, MULTI, CHECK	Dispose, FocusIn, FocusOut, Help, KeyDown, KeyUp, MouseDoubleClick, MouseDown, MouseEnter, MouseExit, MouseHover, MouseUp, MouseMove, Move, Paint, Resize, Selection, DefaultSelection, Collapse, Expand
树项		Dispose

Eclipse 3.0

Eclipse 3.0 还在 SWT 窗口小部件列表中增加了一个浏览器窗口小部件。

参考

Eclipse (O'Reilly) 一书的第 7 章。

8.2 创建一个 SWT 应用程序

问题

你想新建一个 SWT 应用程序。

解决方案

导入 SWT 类, 创建一个 SWT shell, 并将要使用的窗口小部件添加到该 shell 文件中。然后使用该 shell 的 open 方法显示窗口小部件。

讨论

在这个例子中, 我们要创建一个 SWT 窗口, 并在其中显示一些文本。依照例子, 创建一个新的 Java Eclipse 项目, 命名为 *FirstSWTApp*。在 `org.cookbook.ch08` 类中添加一个 `FirstSWTClass` 类。我们将需要导入 SWT 类:

```
package org.cookbook.ch08;

import org.eclipse.swt.widgets.*;
import org.eclipse.swt.*;

.
```

在 main 方法中, 创建一个新的 SWT Display 对象, 并使用该对象创建一个与 SWT 窗口相对应的 Shell 对象。下面是一些最常用的 Shell 方法:

`void addShellListener(ShellListener listener)`

将该监听程序添加到监听程序集合中, 当在 shell 上执行操作时将通知该监听程序。

`void close()`

关闭 shell。

`void dispose()`

处理与该 shell 有关的操作系统资源。

`Rectangle getClientArea()`

返回 shell 的客户区。

`boolean isDisposed()`

如果 shell 已经得到处理, 返回 true, 否则, 返回 false。


```
void open()
```

在屏幕上打开 shell。

```
void setLocation(int x, int y)
```

设置 shell 的位置（以像素为度量单位）。

```
void setText(String s)
```

设置 shell 的标题栏文本。

```
void setSize(int width, int height)
```

设置 shell 的大小。

下面使用 `setText` 和 `setSize` 方法来设置 shell 的标题栏和大小，从而自定义 shell：

```
package org.cookbook.ch08;

import org.eclipse.swt.widgets.*;
import org.eclipse.swt.*;

public class FirstSWTClass {

    public static void main(String [] args) {
        Display display = new Display();
        Shell shell = new Shell(display);
        shell.setText("First SWT Application");
        shell.setSize(250, 250);
        .
        .
        .
    }
}
```

为了显示文本，我们将创建一个 SWT Label 对象，在其中显示文本“Greetings from SWT”。设计标签小部件只是为了显示该文本，可以使用 `setText` 和 `getText` 方法来设置文本。像其他窗口小部件一样，可以使用 `setBounds` 方法来设置标签小部件的边界；在本例中，我们将使标签的尺寸与 shell 的整个客户区一致：

```
package org.cookbook.ch08;

import org.eclipse.swt.widgets.*;
import org.eclipse.swt.*;

public class FirstSWTClass {

    public static void main(String [] args) {
        Display display = new Display();
        Shell shell = new Shell(display);
        shell.setText("First SWT Application");
        shell.setSize(250, 250);
        Label label = new Label(shell, SWT.CENTER);
        label.setText("Greetings from SWT");
        label.setBounds(shell.getClientArea());
    }
}
```

·
·
·

要打开这个窗口, 可调用 `Shell` 对象的 `open` 方法。要管理这个窗口, 可使用 `Shell` 对象的 `isDisposed` 方法来确定该窗口何时已关闭。如果窗口仍然处于打开状态, 可以使用 `Display` 对象的 `readAndDispatch` 方法调用应用程序的消息泵 (message pump)。如果 `shell` 已关闭, 可用 `shell` 的 `dispose` 方法来处理它, 如例 8-3 所示。

注意: 当你使用了 SWT 中的一个资源之后, 应该用 `Widget.dispose` 释放该资源。

例 8-3: FirstSWTClass.java

```
package org.cookbook.ch08;

import org.eclipse.swt.widgets.*;
import org.eclipse.swt.*;

public class FirstSWTClass {

    public static void main(String [] args) {
        Display display = new Display();
        Shell shell = new Shell(display);
        shell.setText("First SWT Application");
        shell.setSize(250, 250);
        Label label = new Label(shell, SWT.CENTER);
        label.setText("Greetings from SWT");
        label.setBounds(shell.getClientArea());
        shell.open();
        while(!shell.isDisposed()) {
            if(!display.readAndDispatch()) display.sleep();
        }
        display.dispose();
    }
}
```

这就是完整的代码, 但如果按照现在的样子输入代码, 将会出现大量弯弯曲曲的红线, 这是因为尚未将 SWT .jar 文件添加到编译路径中。为了使 Eclipse 能够访问所需要的 SWT 类, 请阅读 8.3 节。

参考

8.3 节, 将所需的 SWT JAR 文件添加到编译路径中; 8.4 节, 启动 SWT 应用程序; *Eclipse* (O'Reilly) 一书的第 7 章。

8.3 将所需的 SWT JAR 文件添加到编译路径中

问题

要运行 SWT 应用程序，Eclipse 需要知道到哪里能够找到 SWT 类。

解决方案

将 *swt.jar* 文件添加到编译路径中。

讨论

要使一个 SWT 应用程序（比如上一节创建了其代码的应用程序）能够运行，必须确保导入了以下代码：

```
package org.cookbook.ch08;

import org.eclipse.swt.widgets.*;
import org.eclipse.swt.*;

.
.
.
```

所需要做的就是将 *swt.jar* 文件添加到编译路径中。众所周知，SWT 是操作系统相关的，所以针对不同的操作系统，有不同的 *swt.jar* 文件。

要将 *swt.jar* 文件添加到项目中，可在 Package Explorer 视图中选择项目，右击它，并单击 Properties。在 Properties 对话框中，单击 Java Build Path，并单击 Add External JARs 按钮。然后找到 *swt.jar* 文件。在下列目录中可以找到 *swt.jar*，这取决于具体的操作系统（*HOMEDIR* 是安装 Eclipse 的目录）：

Win32

HOMEDIR\eclipse\plugins\org.eclipse.swt.win32_2.1.2\ws\win32\swt.jar

Linux GTK

HOMEDIR/eclipse/plugins/org.eclipse.swt.gtk_2.1.2/ws/gtk/swt.jar

Linux Motif

HOMEDIR/eclipse/plugins/org.eclipse.swt.motif_2.1.2/ws/motif/swt.jar

Solaris Motif

HOMEDIR/eclipse/plugins/org.eclipse.swt.motif_2.1.2/ws/solaris/sparc/swt.jar

*AIX Motif**HOMEDIR/eclipse/plugins/org.eclipse.swt.motif_2.1.2/ws/aix/ppc/swt.jar**HPUX Motif**HOMEDIR/eclipse/plugins/org.eclipse.swt.motif_2.1.2/ws/hpux/PA_RISC/swt.jar**Photon QNX**HOMEDIR/eclipse/plugins/org.eclipse.swt.photon_2.1.2/ws/photon/swt.jar**Mac OS X**HOMEDIR/eclipse/plugins/org.eclipse.swt.carbon_2.1.2/ws/carbon/swt.jar*

注意：你可能需要针对你的 Eclipse 版本修改这些路径，比如把 2.1.2 改为 2.1.3。Eclipse 3.0 最新的里程碑版本为 3.0.0。

浏览到 swt.jar 文件，单击 Open，然后单击 OK。

注意：有些操作系统，如 Linux GTK，需要的 .jar 文件不止一个；在 Linux GTK 中，需要使用 swt.jar 和 swt-pi.jar 文件。在同一个文件夹中可以找到所需的全部 .jar 文件。

参考

8.2 节，创建一个 SWT 应用程序；8.4 节，启动 SWT 应用程序。

8.4 启动 SWT 应用程序

问题

你想启动 SWT 应用程序。

解决方案

当启动一个 SWT 应用程序时，需要告诉 JVM 在何处可以找到 SWT 的 Java 本机接口 (Java Native Interface, JNI) 代码支持。下面是运行 SWT 应用程序时需要提供的参数（如前所述，*HOMEDIR* 是安装 Eclipse 的目录，而你应该更新这些路径，以便与你的 Eclipse 版本相匹配。例如，2.1.2 应改为 2.1.3 或其他版本号）：

Win32

```
-Djava.library.path=HOMEDIR\eclipse\plugins\org.eclipse.swt.win32_2.1.2\os\win32\x86
```

Linux GTK

```
-Djava.library.path=HOMEDIR/eclipse/plugins/org.eclipse.swt.gtk_2.1.2/os/linux/x86
```

Linux Motif

```
-Djava.library.path=HOMEDIR/eclipse/plugins/org.eclipse.swt.motif_2.1.2/os/linux/x86
```

Solaris Motif

```
-Djava.library.path=HOMEDIR/eclipse/plugins/org.eclipse.swt.motif_2.1.2/os/solaris/sparc
```

AIX Motif

```
-Djava.library.path=HOMEDIR/eclipse/plugins/org.eclipse.swt.motif_2.1.2/os/aix/ppc
```

HPUX Motif

```
-Djava.library.path=HOMEDIR/eclipse/plugins/org.eclipse.swt.motif_2.1.2/os/hpux/PA_RISC
```

Photon QNX

```
-Djava.library.path=HOMEDIR/eclipse/plugins/org.eclipse.swt.photon_2.1.2/os/qnx/x86
```

Mac OS X

```
-Djava.library.path=HOMEDIR/eclipse/plugins/org.eclipse.swt.carbon_2.1.2/os/macosx/ppc
```

讨论

即使在作为 SWT 应用程序编译之后，在准备运行之前，还需要完成另一个步骤。必须向 JVM 传递一个参数，告诉它在何处可以找到 SWT 的本机代码支持。

要启动我们在前两节开发的 SWT 应用程序，可以在 Package Explorer 视图中选择要运行的 FirstSWTClass 类，并选择 Run → Run，以设置启动配置。

在 Launch Configurations 对话框中，选择 Java Application，并单击 New 按钮。Name、Project 和 Main class 文本框应该已经自动填充；如果尚未填充，请填充这些文本框。然

后单击 Arguments 标签。在 VM arguments 文本框中，输入前面提供的 JVM 参数（如图 8-3 所示）。

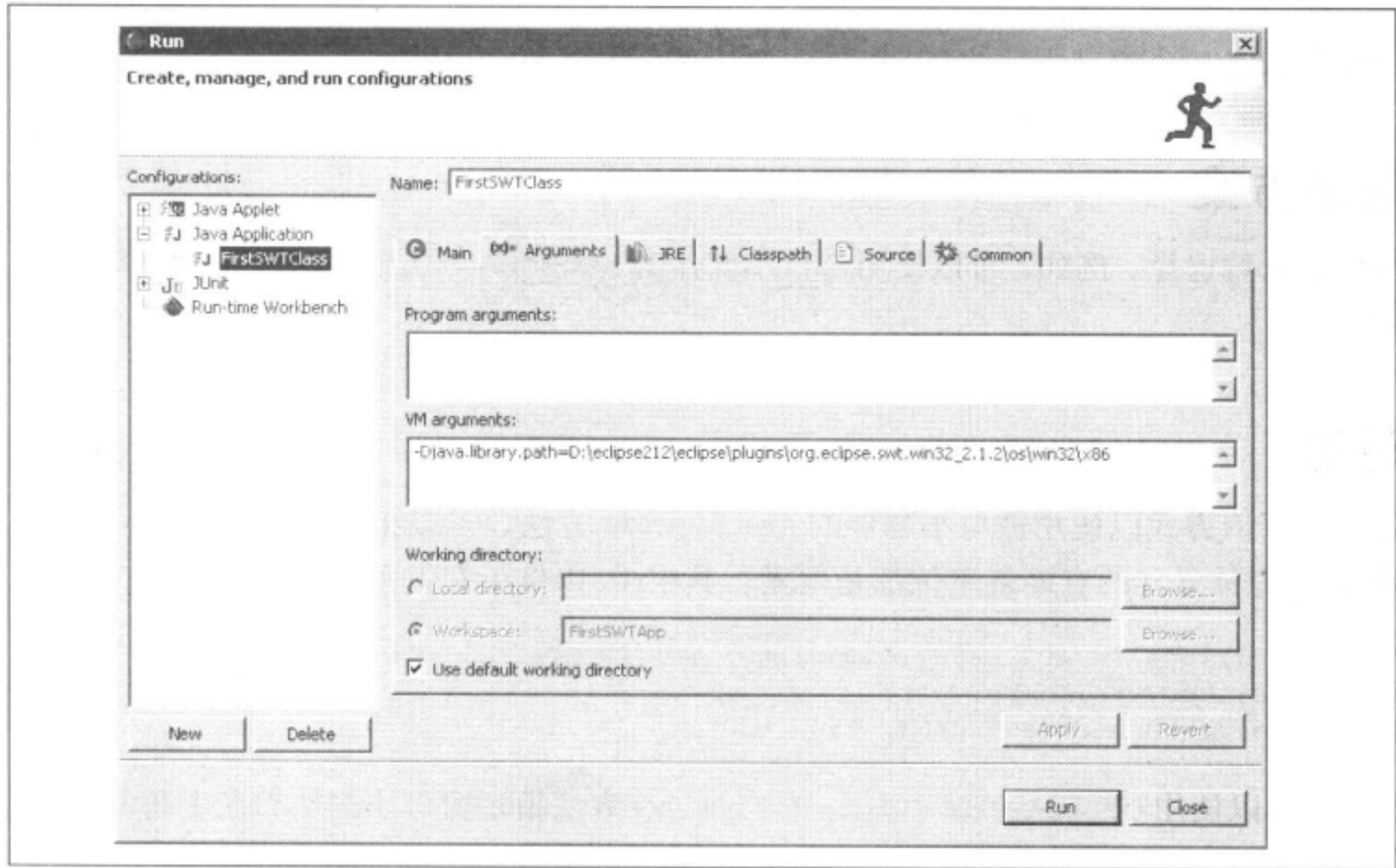


图 8-3: 设置 SWT 应用程序的启动配置

然后单击 Apply 按钮，再单击 Run 按钮。你应该看到，这个新的例子可以运行了，如图 8-4 所示。

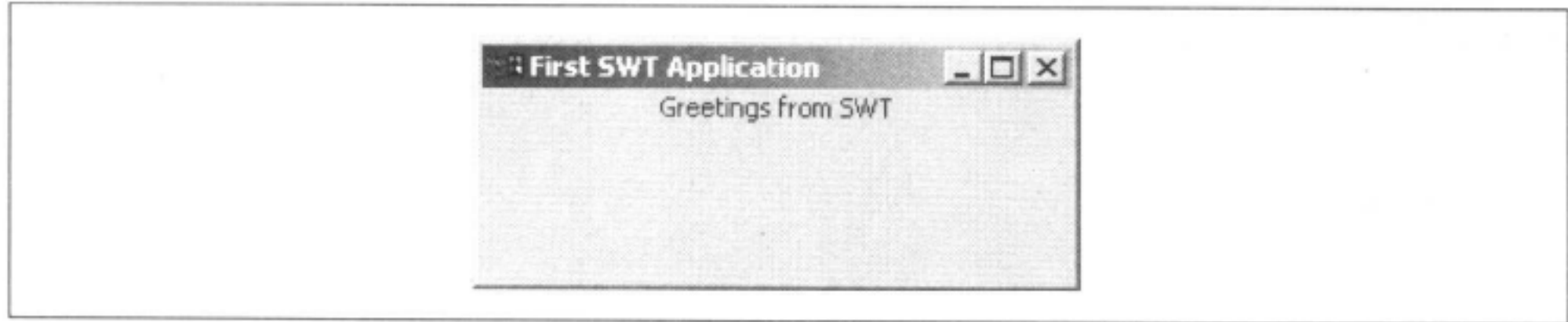


图 8-4: 运行一个 SWT 应用程序

恭喜！你已经成为一名 SWT 开发人员了。

参考

8.2 节，创建一个 SWT 应用程序；8.3 节，将所需的 SWT JAR 文件添加到编译路径中；Eclipse（O'Reilly）一书的第 7 章。

8.5 定位窗口小部件和使用布局

问题

你需要确定一个窗口小部件在 SWT shell 中的位置。

解决方案

可以有几种选择。例如，可以使用窗口小部件的 `setBounds` 方法、其他方法或 SWT 布局。

讨论

通过如下的方式以使用窗口小部件的 `setBounds` 方法，可以指定 SWT 窗口小部件在 shell 中的位置（所有度量单位都是像素，其中(0, 0)位于左上角）：

```
Label label = new Label(composite, SWT.PUSH);
label.setText("Label 0");
label.setBounds(100, 100, 150, 120);
```

另外，可以使用 `setLocation(int x, int y)` 方法指定窗口小部件的左上角的位置，使用 `setSize(int width, int height)` 方法指定窗口小部件的大小。

注意：除了使用 `setSize` 方法以外，还可以使用窗口小部件的 `pack()` 方法调整窗口小部件的大小（例如，对于一个按钮，`pack()` 方法设置按钮的大小，以便容纳按钮的标题）。

还可以使用 SWT 布局来指定窗口小部件的排列方式。SWT 有 4 个内置的布局类：

`FillLayout`

填充 shell 的客户区。

`FormLayout`

布置窗口小部件相对于父复合小部件或另一个窗口小部件的位置。

`GridLayout`

将窗口小部件布置成一个网格。

`RowLayout`

将窗口小部件排列成行或列。

下面是一个使用网格布局的例子。在这个例子中，将创建一个由 4 列组成的网格，并用

SWT 标签填充这些列。首先创建一个 GridLayout 对象，将布局中列的数量设置为 4，并使用 setLayout 将这个布局固定在 shell 中：

```
import org.eclipse.swt.*;
import org.eclipse.swt.layout.*;
import org.eclipse.swt.widgets.*;

public class LayoutClass {

    public static void main (String [] args) {
        Display display = new Display ();
        final Shell shell = new Shell (display);
        shell.setSize(200, 200);
        shell.setText("Layout Example");
        GridLayout gridLayout = new GridLayout();
        gridLayout.numColumns = 4;
        shell.setLayout(gridLayout);
        .
        .
        .
    }
}
```

现在，所需要做的就是创建一组标签，并将它添加到布局中，其中标签将自动排列成一个网格，如例 8-4 所示。

例 8-4: 使用 SWT 布局

```
package org.cookbook.ch08;

import org.eclipse.swt.*;
import org.eclipse.swt.layout.*;
import org.eclipse.swt.widgets.*;

public class LayoutClass {

    public static void main (String [] args) {
        Display display = new Display ();
        final Shell shell = new Shell (display);
        shell.setSize(200, 200);
        shell.setText("Layout Example");
        GridLayout gridLayout = new GridLayout();
        gridLayout.numColumns = 4;
        shell.setLayout(gridLayout);

        for (int loopIndex = 0; loopIndex < 28; loopIndex++) {
            Label label = new Label(shell, SWT.SHADOW_NONE);
            label.setText("Label " + loopIndex);
        }

        shell.open ();
        while (!shell.isDisposed()) {
            if (!display.readAndDispatch()) display.sleep();
        }
        display.dispose ();
    }
}
```



```
}  
}
```

结果如图 8-5 所示，标签排列成一个网格布局。

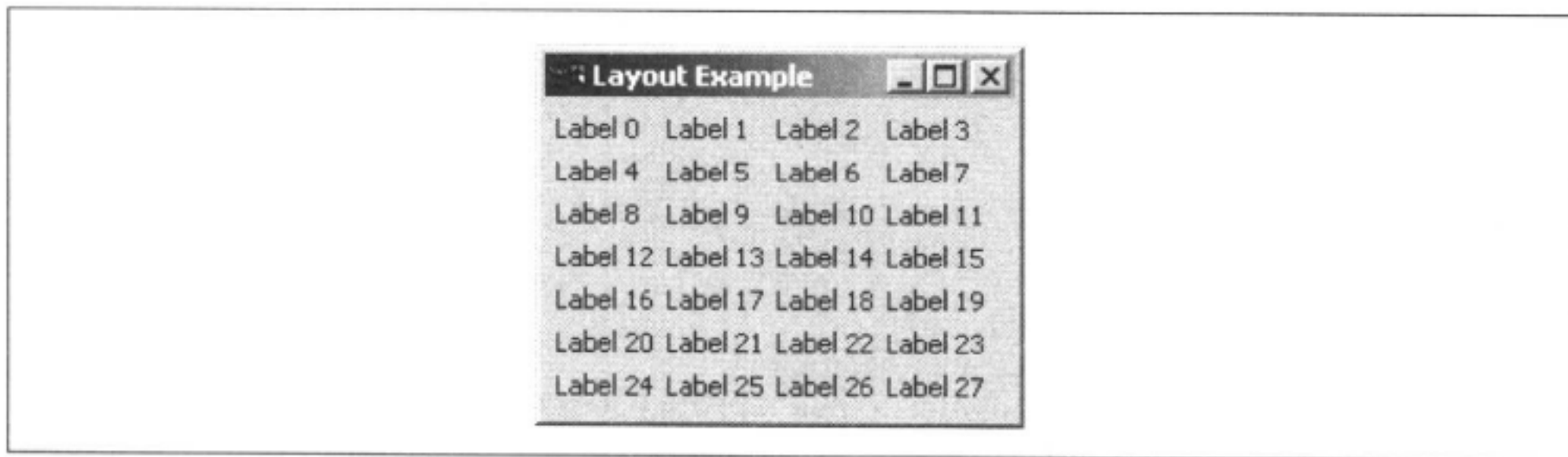


图 8-5：标签排列成一个网格布局

参考

8.6 节，创建按钮和文本小部件；8.9 节，创建复合小部件；8.8 节，创建列表小部件；*Eclipse* (O'Reilly) 一书的第 7 章。

8.6 创建按钮和文本小部件

问题

你需要使用按钮与用户进行交互，并在应用程序中显示文本。

解决方案

使用 SWT 按钮和文本小部件，并使用 SWT 监听程序捕获按钮事件。

讨论

下面这个例子将在 shell 中添加 SWT 按钮小部件，捕获单击事件，并在按钮被单击时显示一条文本消息。首先，在新项目 *ButtonApp* 中创建一个按钮小部件。下面是节选的一些最常用的按钮小部件方法：

```
void addSelectionListener(SelectionListener listener)
```

将该监听程序添加到监听程序集合中，当按钮被单击时将通知该监听程序。

String getText()

返回按钮的标题。

void setBounds(int x, int y, int width, int height)

根据参数指定的区域设置按钮的大小和位置。

void setImage(Image image)

设置按钮的图像。

void setText(String string)

设置按钮的标题。

创建按钮小部件相当容易；只需使用Button类的构造函数，用setBounds方法或布局定位按钮，并用setText设置按钮的标题即可：

```
package org.cookbook.ch08;

import org.eclipse.swt.widgets.*;
import org.eclipse.swt.SWT;
import org.eclipse.swt.events.*;

public class ButtonClass {

    public static void main(String [] args) {
        Display display = new Display();
        Shell shell = new Shell(display);
        shell.setSize(200, 200);
        shell.setText("Button Example");

        final Button button = new Button(shell, SWT.PUSH);
        button.setBounds(40, 50, 50, 20);
        button.setText("Click Me");
        .
        .
        .
    }
}
```

创建一个文本小部件同样简单。下面是节选的一些常用的文本小部件方法：

void copy()

复制选定的文本。

void cut()

剪切选定的文本。

String getSelectionText()

获得选定的文本。

String getText()

获得窗口小部件的文本。

`void paste()`

从剪贴板粘贴文本。

`void setBounds(int x, int y, int width, int height)`

根据参数指定的区域设置文本小部件的大小和位置。

`void setFont(Font font)`

设置字体。

`void setText(String string)`

设置文本小部件的内容。

在下面这个例子中，将设置文本小部件的边框，并将它添加到 shell 中：

```
public class ButtonClass {  
    public static void main(String [] args) {  
        Display display = new Display();  
        Shell shell = new Shell(display);  
        shell.setSize(200, 200);  
        shell.setText("Button Example");  
  
        final Button button = new Button(shell, SWT.PUSH);  
        button.setBounds(40, 50, 50, 20);  
        button.setText("Click Me");  
  
        final Text text = new Text(shell, SWT.BORDER);  
        text.setBounds(100, 50, 70, 20);  
        .  
        .  
        .  
    }  
}
```

上述代码确定了按钮和文本小部件的位置；下一步是处理按钮事件，这将在 8.7 节中讨论。

参考

8.1 节，使用 SWT 窗口小部件；8.7 节，处理 SWT 窗口小部件事件；*Eclipse* (O'Reilly) 一书的第 7 章。

8.7 处理 SWT 窗口小部件事件

问题

你需要捕获窗口小部件事件（如按钮单击），并通过代码响应这些事件。

解决方案

使用 SWT 监听程序。在 SWT 中，监听程序与 AWT 中的监听程序非常相似。下面是一些最常用的 SWT 监听程序：

`ControlListener`

处理移动和大小调整事件。

`FocusListener`

处理获得和失去焦点事件。

`KeyListener`

处理键击事件。

`MouseListener`, `MouseMoveListener`, `MouseTrackListener`

处理鼠标事件。

`SelectionListener`

处理窗口小部件选定事件（包括按钮单击）。

讨论

为了弄清如何处理窗口小部件事件，我们将继续 8.6 节的例子，在这个例子中，我们需要在单击按钮时在文本小部件中显示文本。使用 `addSelectionListener` 方法将 `SelectionListener` 对象添加到按钮中，即可捕获按钮单击事件。要实现 `SelectionListener` 接口，必须实现两个方法：`widgetSelected` 和 `widgetDefaultSelected`，前者当一个窗口小部件中有文本被选中时发生，后者当在一个窗口小部件中进行默认选择时发生。

在本例中，使用一个匿名内部类中的 `SelectionListener` 对象，在文本小部件中显示文本 “No problem”：

```
public class ButtonClass {  
    public static void main(String [] args) {  
        Display display = new Display();  
        Shell shell = new Shell(display);  
        shell.setSize(200, 200);  
        shell.setText("Button Example");  
  
        final Button button = new Button(shell, SWT.PUSH);  
        button.setBounds(40, 50, 50, 20);  
        button.setText("Click Me");  
  
        final Text text = new Text(shell, SWT.BORDER);  
        text.setBounds(100, 50, 70, 20);  
    }  
}
```



```

button.addSelectionListener(new SelectionListener()
{
    public void widgetSelected(SelectionEvent event)
    {
        text.setText("No problem");
    }

    public void widgetDefaultSelected(SelectionEvent event)
    {
        text.setText("No worries!");
    }
});
.
.
.

```

注意：对于每一个监听程序类，都有可用的适配器类，可以根据一个适配器类来扩展监听程序类，而不必实现监听程序类的接口。这样做可以省去实现监听程序类的所有方法的麻烦。例如，SelectionListener接口的适配器类被命名为SelectionAdapter。关于如何扩展监听程序类的具体例子，请参阅9.8节。

在这个例子中，需要补充的就是在必要时关闭 shell 的代码（见例 8-5）。

例 8-5：使用 SWT 按钮和文本小部件

```

package org.cookbook.ch08;

import org.eclipse.swt.widgets.*;
import org.eclipse.swt.SWT;
import org.eclipse.swt.events.*;

public class ButtonClass {

    public static void main(String [] args) {
        Display display = new Display();
        Shell shell = new Shell(display);
        shell.setSize(200, 200);
        shell.setText("Button Example");

        final Button button = new Button(shell, SWT.PUSH);
        button.setBounds(40, 50, 50, 20);
        button.setText("Click Me");

        final Text text = new Text(shell, SWT.BORDER);
        text.setBounds(100, 50, 70, 20);

        button.addSelectionListener(new SelectionListener()
        {
            public void widgetSelected(SelectionEvent event)
            {

```

```
        text.setText("No problem");
    }

    public void widgetDefaultSelected(SelectionEvent event)
    {
        text.setText("No worries!");
    }
});

shell.open();
while(!shell.isDisposed()) {
    if(!display.readAndDispatch()) display.sleep();
}
display.dispose();
}
```

运行结果如图 8-6 所示；当单击按钮时，文本消息“No problem”将出现在文本小部件中。

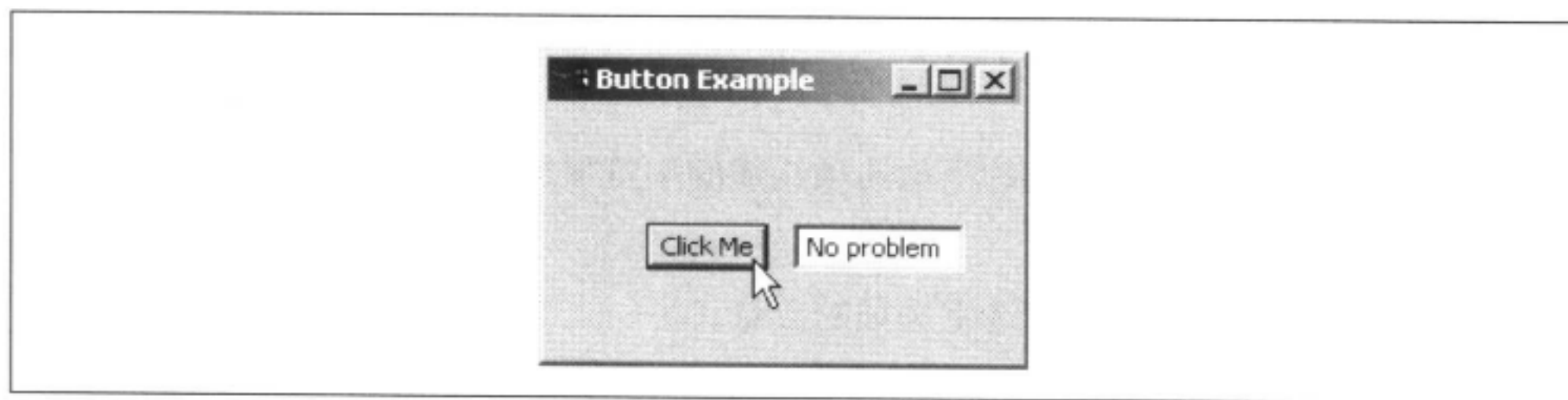


图 8-6：使用按钮和文本小部件

参考

8.1 节，使用 SWT 窗口小部件；8.6 节，创建按钮和文本小部件。

8.8 创建列表小部件

问题

你需要显示多个选项，以便用户从中选择。

解决方案

使用列表小部件，它可以显示选项的列表。下面是节选的一些最常用的列表小部件方法：

```
void add(String string)
```

将参数添加到列表小部件的列表末尾。

```
void add(String string, int index)
```

将参数添加到列表小部件的列表中给定零相对索引的位置。

```
void addSelectionListener(SelectionListener listener)
```

将该监听程序添加到监听程序集合中,当列表小部件中的选择发生变化时将通知该监听程序。

```
String getItem(int index)
```

返回列表小部件中给定零相对索引的选项。

```
int getItemCount()
```

返回列表小部件中包含选项的数量。

```
String[] getItems()
```

返回一个由列表小部件中的选项构成的字符串数组。

```
String[] getSelection()
```

返回一个由列表小部件中的当前选项构成的字符串数组。

```
int getSelectionCount()
```

返回列表小部件中包含的选定选项的数量。

```
int getSelectionIndex()
```

返回列表小部件中当前选定选项的零相对索引;如果没有选定任何选项,则返回-1。

```
int[] getSelectionIndices()
```

返回列表小部件中当前选定的所有选项的零相对索引。

```
void setItems(String[] items)
```

设置列表小部件的选项,使其成为特定数组的元素。

```
void setSelection(int index)
```

选择列表小部件中给定零相对索引处的选项。

```
void setSelection(int[] indices)
```

选择列表小部件中所有给定零相对索引处的选项

```
void setSelection(String[] items)
```

设置列表小部件的选择,使其成为特定数组的元素。

讨论

作为一个例子,我们将使用`getSelectionIndices`方法在`shell`中添加一个多选列表小部件,并获得用户的选择;这个例子被命名为`ListApp`,在本书的站点可以找到这个例子。下面介绍如何将列表小部件添加到`shell`中:

```
import org.eclipse.swt.*;
import org.eclipse.swt.widgets.*;
import org.eclipse.swt.events.*;

public class ListClass {

    public static void main (String [] args) {
        Display display = new Display ();
        Shell shell = new Shell (display);
        shell.setText("List Example");
        shell.setSize(300, 200);

        final List list = new List (shell, SWT.BORDER | SWT.MULTI | SWT.V_SCROLL);
        list.setBounds(40, 20, 220, 100);
        for (int loopIndex = 0; loopIndex < 9; loopIndex++){
            list.add("Item Number " + loopIndex);
        }
        .
        .
        .
    }
}
```

可以在这个列表小部件中添加一个选择监听程序,如例8-6所示。在这个例子中,我们将使用列表小部件的`getSelectionIndices`方法来获得由列表小部件中选定索引构成的`int`数组。正如在代码中看到的,我们将在一个文本小部件中显示这些选项。

例8-6: 使用列表小部件

```
package org.cookbook.ch08;

import org.eclipse.swt.*;
import org.eclipse.swt.widgets.*;
import org.eclipse.swt.events.*;

public class ListClass {

    public static void main (String [] args) {
        Display display = new Display ();
        Shell shell = new Shell (display);
        shell.setText("List Example");
        shell.setSize(300, 200);

        final List list = new List (shell, SWT.BORDER | SWT.MULTI | SWT.V_SCROLL);
        list.setBounds(40, 20, 220, 100);
        for (int loopIndex = 0; loopIndex < 9; loopIndex++){
            list.add("Item Number " + loopIndex);
        }
    }
}
```



```
final Text text = new Text(shell, SWT.BORDER);
text.setBounds(60, 130, 160, 25);

list.addSelectionListener(new SelectionListener()
{
    public void widgetSelected(SelectionEvent event)
    {
        int [] selectedItems = list.getSelectionIndices ();
        String outString = "";
        for (int loopIndex = 0; loopIndex < selectedItems.length;
            loopIndex++) outString += selectedItems[loopIndex] + " ";
        text.setText("Selected Items: " + outString);
    }

    public void widgetDefaultSelected(SelectionEvent event)
    {
        int [] selectedItems = list.getSelectionIndices ();
        String outString = "";
        for (int loopIndex = 0; loopIndex < selectedItems.length; loopIndex++)
            outString += selectedItems[loopIndex] + " ";
        System.out.println ("Selected Items: " + outString);
    }
});

shell.open ();
while (!shell.isDisposed ()) {
    if (!display.readAndDispatch ()) display.sleep ();
}
display.dispose ();
}
```

运行结果如图 8-7 所示，其中显示了用户已经做出的选择。

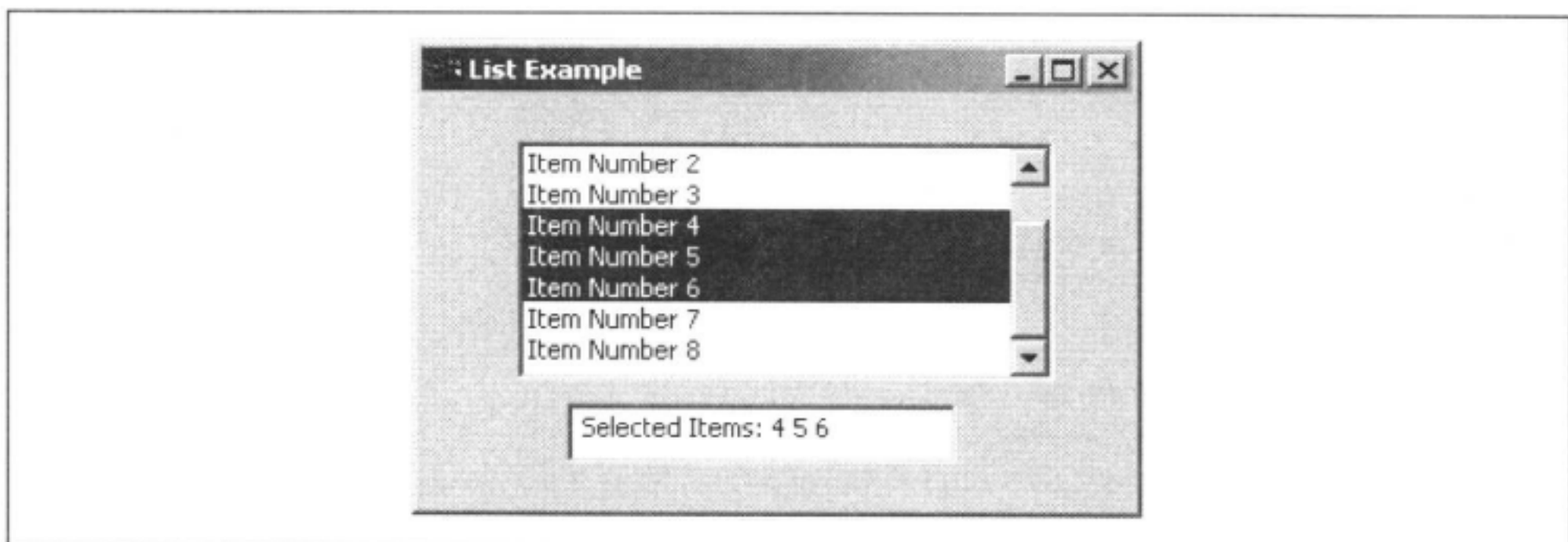


图 8-7：使用列表小部件

8.9 创建复合小部件

问题

要创建自定义的窗口小部件，需要创建一个包含其他窗口小部件的窗口小部件。

解决方案

使用容器小部件，如复合小部件，从 Composite 类内部创建一个对象。然后将窗口小部件添加到复合对象中，就像在 Shell 对象中添加窗口小部件一样。

讨论

大量的窗口小部件，如复合小部件，被设计用来容纳其他小部件，从而使你能够构建复合小部件。例 8-7 显示了一个复合小部件（本书站点的 CompositeApp 示例）。在本例中，将显示与使用网格布局显示的相同标签，但是在一个复合小部件内的 shell 中。首先创建一个复合小部件，并在其中放置一个网格布局：

```
final Composite composite = new Composite(shell, SWT.NONE);
GridLayout gridLayout = new GridLayout();
gridLayout.numColumns = 4;
composite.setLayout(gridLayout);
```

现在，把这个复合小部件看作是一个容器，就像一个 shell，并在其中添加标签小部件，如例 8-7 所示。

例 8-7: 使用复合小部件

```
package org.cookbook.ch08;

import org.eclipse.swt.*;
import org.eclipse.swt.layout.*;
import org.eclipse.swt.widgets.*;

public class CompositeClass {

    public static void main (String [] args) {
        Display display = new Display ();
        final Shell shell = new Shell (display);
        shell.setSize(300, 300);
        shell.setLayout(new RowLayout());

        shell.setText("Composite Example");

        final Composite composite = new Composite(shell, SWT.NONE);
        GridLayout gridLayout = new GridLayout();
        gridLayout.numColumns = 4;
        composite.setLayout(gridLayout);
```

```
        for (int loopIndex = 0; loopIndex < 28; loopIndex++) {  
            Label label = new Label(composite, SWT.SHADOW_NONE);  
            label.setText("Label " + loopIndex);  
        }  
  
        shell.open ();  
        while (!shell.isDisposed()) {  
            if (!display.readAndDispatch()) display.sleep();  
        }  
        display.dispose ();  
    }  
}
```

容纳标签的复合小部件出现在 shell 中，如图 8-8 所示。

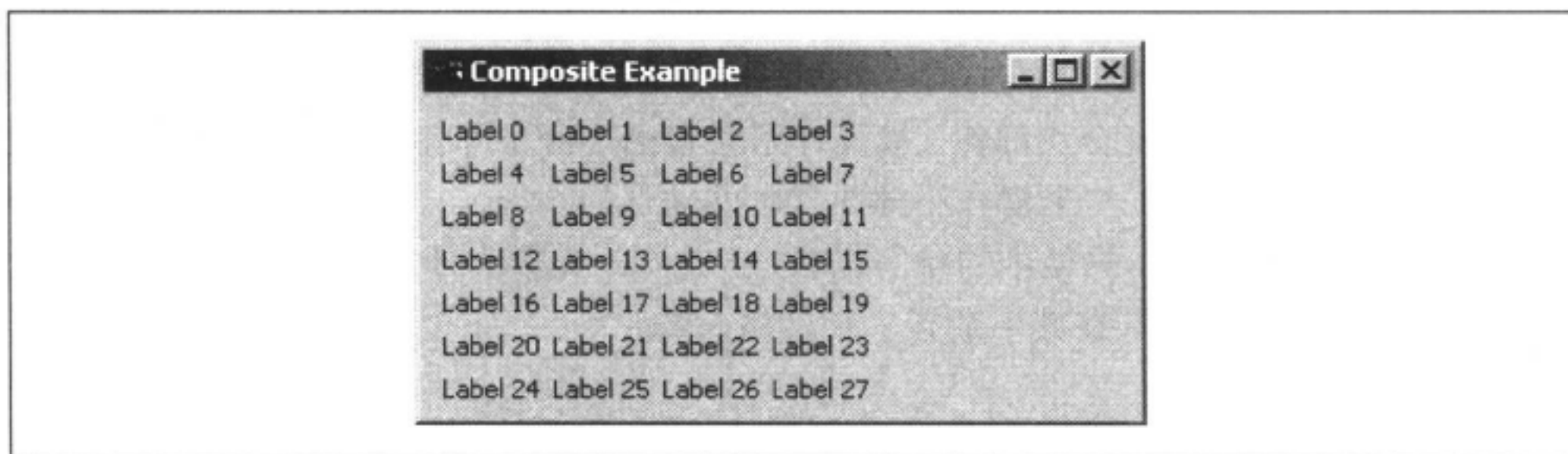


图 8-8：使用复合小部件

除了复合小部件以外，可以容纳窗口小部件的其他窗口小部件包括画布（设计用来绘图）和组（也可以显示边框）。

8.10 创建非矩形窗口

问题

你想用异形窗口令用户拍案叫绝，这些窗口用来显示公司徽标、棋子、广告等。

解决方案

使用 Eclipse 3.0 中的 shell 区域创建一个非矩形 shell。在 Eclipse 3.0 中，SWT shell 现在可以采用按区域定义的不规则形状。作为一个例子（本书站点的 *NonRectangularApp* 示例），我们将创建一个蓝色的面包圈形状的窗口，其中有一个 Exit 按钮。所需要做的就是创建一个 shell，创建定义要使用形状的 Region 对象，并使用该 shell 的 `setRegion` 方法配置该 shell。

讨论

在这个例子中，我们将使用一个名为 `createCircle` 的方法创建圆形区域。下面的示例说明了如何使用这个方法及其创建的圆形区域，来创建面包圈形状的 `shell`，并使用该 `shell` 的 `setBackground` 方法将其涂成蓝色。注意，使用 `Region` 类的 `add` 和 `subtract` 方法，可以在最终的 `shell` 中增加或剪切一些区域：

```
final Display display = new Display();
final Shell shell = new Shell(display, SWT.NO_TRIM);

Region region = new Region();
region.add(createCircle(50, 50, 50));
region.subtract(createCircle(50, 50, 20));
shell.setRegion(region);
shell.setSize(region.getBounds().width, region.getBounds().height);
shell.setBackground(display.getSystemColor(SWT.COLOR_BLUE));
.
.
.
```

为了创建圆，`createCircle` 方法返回一个由圆周上的点构成的数组。下面的代码说明了这个方法是如何完成这项任务的：

```
static int[] createCircle(int xOffset, int yOffset, int radius) {
    int[] circlePoints = new int[10 * radius];
    for (int loopIndex = 0; loopIndex < 2 * radius + 1; loopIndex++) {
        int xCurrent = loopIndex - radius;
        int yCurrent = (int) Math.sqrt(radius * radius
            - xCurrent * xCurrent);
        int doubleLoopIndex = 2 * loopIndex;

        circlePoints[doubleLoopIndex] = xCurrent + xOffset;
        circlePoints[doubleLoopIndex + 1] = yCurrent + yOffset;
        circlePoints[10 * radius - doubleLoopIndex - 2] = xCurrent + xOffset;
        circlePoints[10 * radius - doubleLoopIndex - 1] = -yCurrent
            + yOffset;
    }

    return circlePoints;
}
```

剩下的工作就是创建一个 `Exit` 按钮，并将它添加到代码中，如例 8-8 所示。

例 8-8: 创建非矩形窗口

```
package org.cookbook.ch08;

import org.eclipse.swt.*;
import org.eclipse.swt.graphics.*;
import org.eclipse.swt.widgets.*;
```



```

public class NonRectangularClass {

    static int[] createCircle(int xOffset, int yOffset, int radius) {
        int[] circlePoints = new int[10 * radius];
        for (int loopIndex = 0; loopIndex < 2 * radius + 1; loopIndex++) {
            int xCurrent = loopIndex - radius;
            int yCurrent = (int) Math.sqrt(radius * radius - xCurrent
                * xCurrent);
            int doubleLoopIndex = 2 * loopIndex;

            circlePoints[doubleLoopIndex] = xCurrent + xOffset;
            circlePoints[doubleLoopIndex + 1] = yCurrent + yOffset;
            circlePoints[10 * radius - doubleLoopIndex - 2] = xCurrent + xOffset;
            circlePoints[10 * radius - doubleLoopIndex - 1] = -yCurrent
                + yOffset;
        }

        return circlePoints;
    }

    public static void main(String[] args) {
        final Display display = new Display();
        final Shell shell = new Shell(display, SWT.NO_TRIM);

        Region region = new Region();
        region.add(createCircle(50, 50, 50));
        region.subtract(createCircle(50, 50, 20));
        shell.setRegion(region);
        shell.setSize(region.getBounds().width, region.getBounds().height);
        shell.setBackground(display.getSystemColor(SWT.COLOR_BLUE));

        Button button = new Button(shell, SWT.PUSH);
        button.setText("Exit");
        button.setBounds(35, 6, 35, 20);
        button.addListener(SWT.Selection, new Listener() {
            public void handleEvent(Event event) {
                shell.close();
            }
        });

        shell.open();
        while (!shell.isDisposed()) {
            if (!display.readAndDispatch())
                display.sleep();
        }
        region.dispose();
        display.dispose();
    }
}

```

这就是所需要全部代码，其运行结果如图 8-9 所示。棒极了！

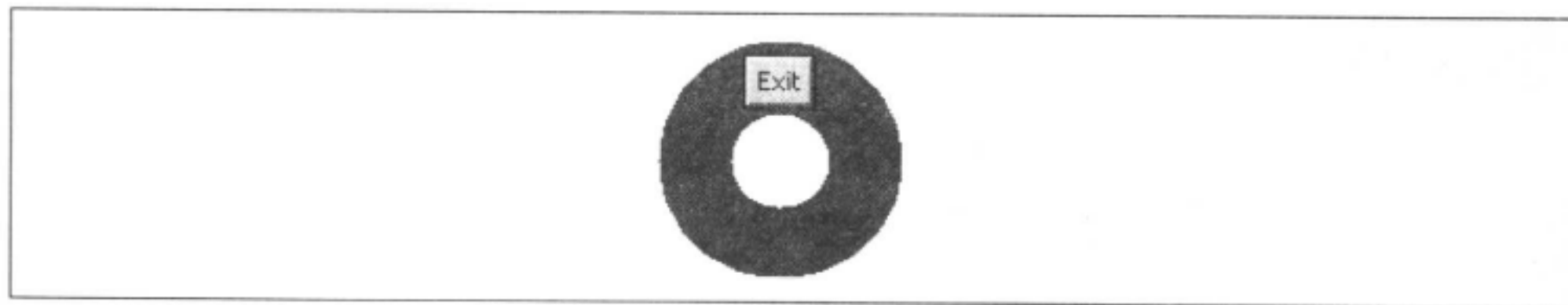


图 8-9: 面包圈形状的窗口

8.11 多线程 SWT 应用程序

问题

你正在一个工人线程中运行代码,同时你需要在SWT应用程序中与用户界面进行交互。

解决方案

使用 `Display.getDefault` 方法访问主 UI 线程。然后使用 `asyncExec` 或 `syncExec` 方法,并向这两个方法传递一个可与用户界面元素交互的 `Runnable` 对象。

讨论

在 SWT 应用程序中,主线程称为 UI 线程,是负责处理事件并把事件派发给窗口小部件的线程。在其他的 GUI 框架中,如在 AWT 或 Swing 中,不必直接与 UI 线程交互,但 SWT 体系结构与众不同 (SWT 使用消息泵体系结构来支持 Eclipse 中的插件)。

如果有大量线程密集型工作要做,由 UI 线程来负担这些工作不是一个好主意,而你可能需要启动一个工人线程。另一方面,UI 线程是唯一能够与 SWT 用户界面元素交互且不会引发异常的线程。

那么,如何从工人线程与用户界面元素进行交互呢? 可以使用 UI 线程的 `asyncExec` 和 `syncExec` 方法,并向这两个方法传递一个可与所需用户界面元素交互的 `Runnable` 对象。下面是一个例子。在这个工人线程代码中,需要在一个文本小部件中显示一些文本。为此,可使用 `Display.getDefault` 方法获得当前的 `Display` 对象,并按照如下方式设置文本小部件中的文本:

```
Display.getDefault().asyncExec(new Runnable()
{
    public void run()
    {
        textWidget.setText("Worker thread task has finished.");
    }
});
```

第 9 章

SWT：对话框、 工具栏及菜单等

9.0 简介

本章继续介绍 SWT，并举例说明如何创建对话框、菜单、表格和工具栏。所有这些都是 SWT 的功能强大的控件，它们比上一章介绍的 SWT 控件要复杂一些，值得详细介绍。本章还将介绍 Eclipse 3.0 中新增的对在 SWT 中嵌入 Swing 和 AWT 元素的支持（如果在 SWT 中找不到所需的控件，就从 Swing 中嵌入它！）。

9.1 创建消息框

问题

你想发送一条消息给用户，并 / 或从用户那里得到一些反馈。

解决方案

使用 SWT 的 `MessageBox` 类创建一个消息框，并检查其 `open` 方法的返回值，以确定用户单击了哪一个按钮。

讨论

下面是 `MessageBox` 类的方法：

```
String getMessage()  
    返回对话框的消息。
```

```
int open()
```

使对话框可见。

```
void setMessage(String string)
```

设置对话框的消息。

通过把当前 shell 以及要使用的样式传递给 `MessageBox` 构造函数，创建一个消息框：`MessageBox(Shell parent, int style)`。下面是一些常用的样式：

- `SWT.ICON_ERROR`
- `SWT.ICON_INFORMATION`
- `SWT.ICON_QUESTION`
- `SWT.ICON_WARNING`
- `SWT.ICON_WORKING`
- `SWT.OK`
- `SWT.OK | SWT.CANCEL`
- `SWT.YES | SWT.NO`
- `SWT.YES | SWT.NO | SWT.CANCEL`
- `SWT.RETRY | SWT.CANCEL`
- `SWT.ABORT | SWT.RETRY | SWT.IGNORE.`

使用消息框的 `open` 方法可以显示消息；当消息框关闭时，将此方法的 `int` 返回值与上述样式（如 `SWT.YES`、`SWT.NO` 或 `SWT.CANCEL`）之一进行对比，以确定单击了哪一个按钮。

9.2 创建对话框

问题

你需要从用户那里获得一些输入，而对话框是最理想的手段。

解决方案

新建一个 shell，把它配置为一个对话框，显示它，并取回用户输入的数据。

讨论

SWT附带有消息框和大量的预制对话框,如创建文件对话框的`FileDialog`类或使用户能够选择字体的`FontDialog`类。但是,如果预制对话框不能满足需要,可以构建自定义的对话框。在下面这个例子中,假设你想确认用户确实要删除一个文件。当用户单击标有Delete File的按钮时,应用程序(本书站点的`DialogApp`项目)显示一个确认对话框,其中包含有OK按钮和Cancel按钮。然后确定用户单击了哪一个按钮。

应用程序首先创建一个shell,其中包含Delete File按钮和一个文本小部件,文本小部件中将显示对用户的按钮单击事件的响应:

```
Display display = new Display();
Shell shell = new Shell(display);
shell.setText("Dialog Example");
shell.setSize(300, 200);
shell.open();

final Button button = new Button(shell, SWT.PUSH);
button.setText("Delete File");
button.setBounds(20, 40, 80, 25);

final Text text = new Text(shell, SWT.SHADOW_IN);
text.setBounds(140, 40, 100, 25);
.
.
.
```

要创建一个对话框,可创建一个shell,并将其样式指定为`SWT.APPLICATION_MODAL | SWT.DIALOG_TRIM`,使其成为一个应用程序模式对话框(在继续使用语言程序之前必须关闭对话框),并将该shell的样式设计成为一个对话框。另外,在对话框中添加一个提示“Delete the file?”以及OK和Cancel按钮:

```
final Shell dialog = new Shell(shell, SWT.APPLICATION_MODAL |
    SWT.DIALOG_TRIM);
dialog.setText("Delete File");
dialog.setSize(250, 150);

final Button buttonOK = new Button(dialog, SWT.PUSH);
buttonOK.setText("OK");
buttonOK.setBounds(20, 55, 80, 25);

Button buttonCancel = new Button(dialog, SWT.PUSH);
buttonCancel.setText("Cancel");
buttonCancel.setBounds(120, 55, 80, 25);

final Label label = new Label(dialog, SWT.NONE);
label.setText("Delete the file?");
label.setBounds(20, 15, 100, 20);
```

上述代码将生成一个对话框，并显示它，你所需要做的就是调用它的open方法。那么，如何判断用户单击了哪一个按钮呢？

为了获得该对话框中的数据，可以使用一个基于匿名内部类的监听程序。为了从监听程序的方法中获取数据，可使用一个类字段。在这个例子中，设置一个布尔变量deleteFlag，在OK按钮的监听程序代码中指示用户是否单击了OK按钮：

```
package org.cookbook.ch09;

import org.eclipse.swt.*;
import org.eclipse.swt.widgets.*;

public class DialogClass {
    static boolean deleteFlag = false;
    .
    .
    .
    Listener listener = new Listener() {
        public void handleEvent(Event event) {
            if(event.widget == buttonOK){
                deleteFlag = true;
            }else{
                deleteFlag = false;
            }
            dialog.close();
        }
    };

    buttonOK.addListener(SWT.Selection, listener);
    buttonCancel.addListener(SWT.Selection, listener);
    .
    .
    .
}
```

余下的工作就是在文本小部件中显示一条适当的消息，消息的内容取决于deleteFlag的值，而这相当简单，如例9-1所示。

例9-1：创建对话框

```
package org.cookbook.ch09;

import org.eclipse.swt.*;
import org.eclipse.swt.widgets.*;

public class DialogClass {
    static boolean deleteFlag = false;

    public static void main(String [] args) {
        Display display = new Display();
        Shell shell = new Shell(display);
        shell.setText("Dialog Example");
        shell.setSize(300, 200);
        shell.open();
    }
}
```

```
final Button button = new Button(shell, SWT.PUSH);
button.setText("Delete File");
button.setBounds(20, 40, 80, 25);

final Text text = new Text(shell, SWT.SHADOW_IN);
text.setBounds(140, 40, 100, 25);

final Shell dialog = new Shell(shell, SWT.APPLICATION_MODAL |
    SWT.DIALOG_TRIM);
dialog.setText("Delete File");
dialog.setSize(250, 150);

final Button buttonOK = new Button(dialog, SWT.PUSH);
buttonOK.setText("OK");
buttonOK.setBounds(20, 55, 80, 25);

Button buttonCancel = new Button(dialog, SWT.PUSH);
buttonCancel.setText("Cancel");
buttonCancel.setBounds(120, 55, 80, 25);

final Label label = new Label(dialog, SWT.NONE);
label.setText("Delete the file?");
label.setBounds(20, 15, 100, 20);

Listener listener = new Listener() {
    public void handleEvent(Event event) {
        if(event.widget == buttonOK){
            deleteFlag = true;
        }else{
            deleteFlag = false;
        }
        dialog.close();
    }
};

buttonOK.addListener(SWT.Selection, listener);
buttonCancel.addListener(SWT.Selection, listener);

Listener buttonListener = new Listener() {
    public void handleEvent(Event event) {
        dialog.open();
    }
};

button.addListener(SWT.Selection, buttonListener);

while(!dialog.isDisposed()) {
    if(!display.readAndDispatch()) display.sleep();
}

if(deleteFlag){
    text.setText("File deleted.");
} else {
    text.setText("File not deleted.");
}
```

```
while(!shell.isDisposed()) {  
    if(!display.readAndDispatch()) display.sleep();  
}  
display.dispose();  
}
```

现在，运行这个示例，并单击窗口中出现的 Delete File 按钮，如图 9-1 所示。

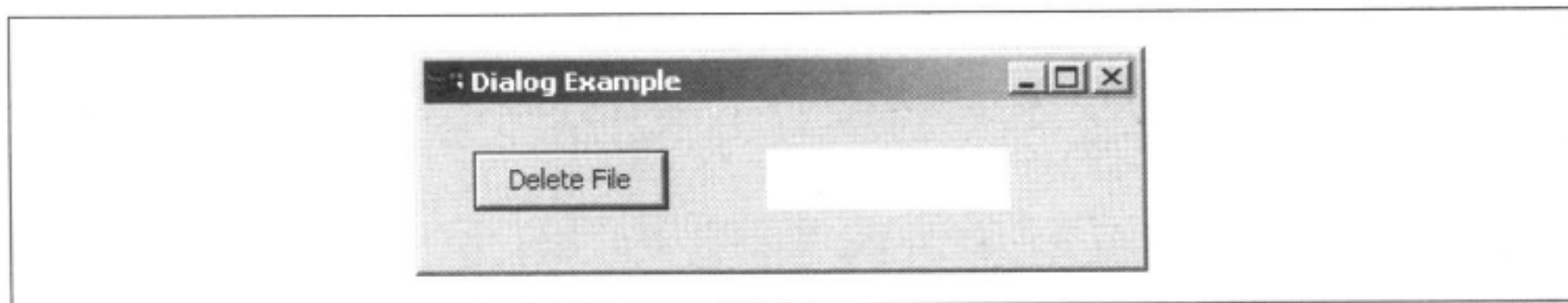


图 9-1: 对话框示例

单击 Delete File 按钮，使对话框出现，如图 9-2 所示。

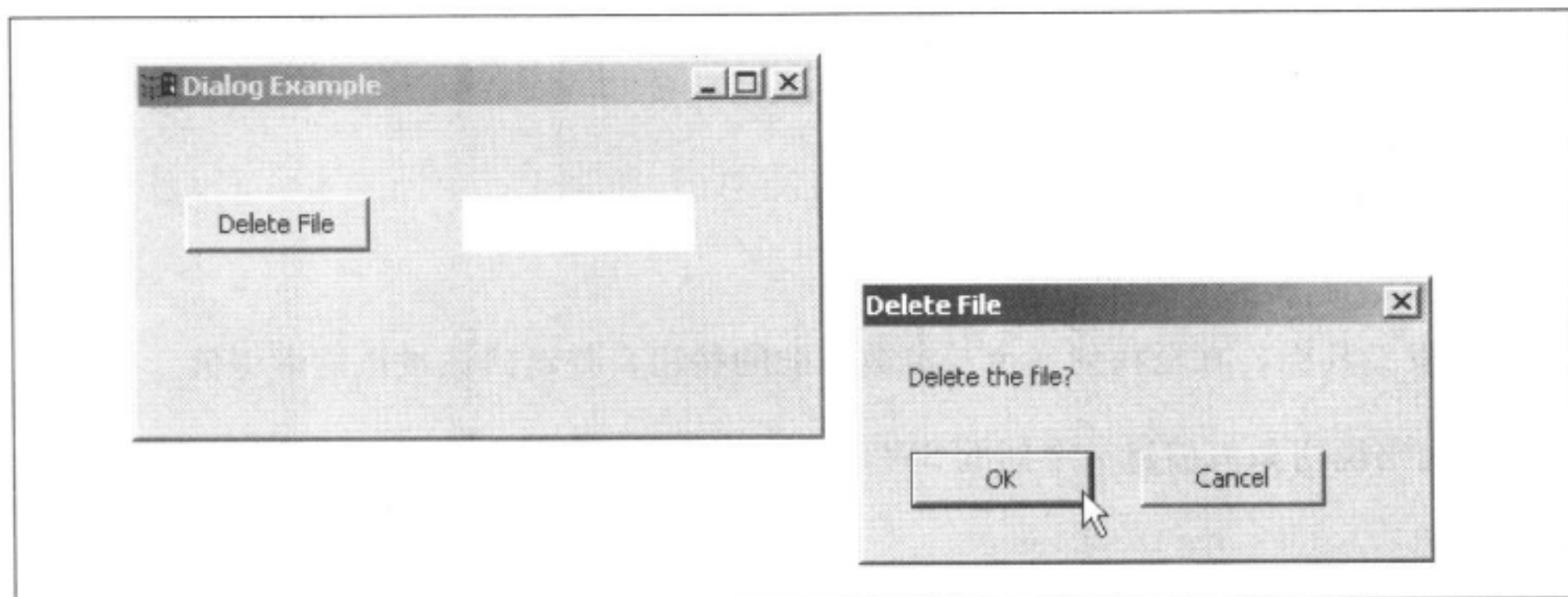


图 9-2: 新建的对话框

现在，单击 OK 按钮，主窗口的文本小部件中将显示一条确认消息，如图 9-3 所示。

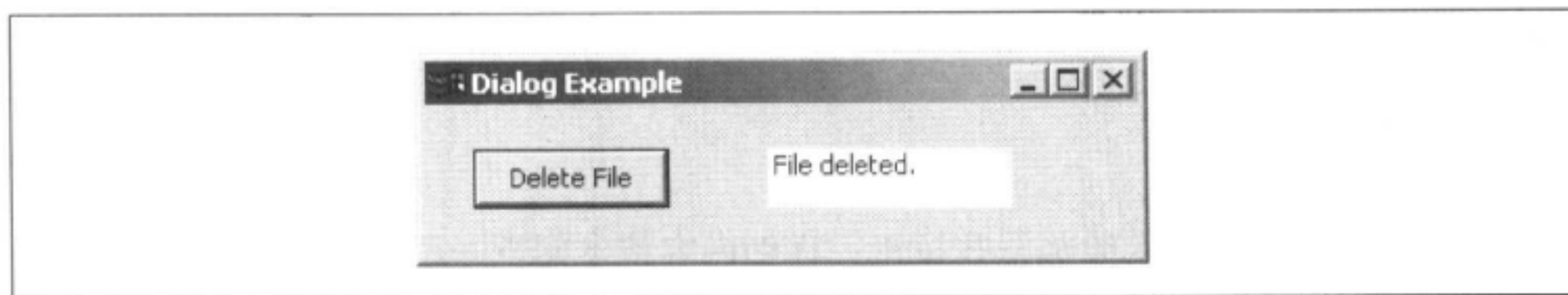


图 9-3: 获取对话框中的结果

9.3 创建工具栏

问题

你想创建一个工具栏，并处理工具栏按钮单击事件。

解决方案

使用 SWT 的 `ToolBar` 和 `ToolItem` 类。

讨论

我们将在一个新项目（本书站点的 *ToolBarApp* 项目）中创建一个示例工具栏，并在其上放置工具栏按钮，同时报告用户单击了哪一个按钮。要创建工具栏，需要使用 SWT 的 `ToolBar` 类；下面是节选的一些此类的最常用的方法：

```
int getItemCount()
```

返回工具栏上包含的按钮的数量。

```
ToolItem[] getItems()
```

返回一个由工具栏按钮构成的数组。

```
int indexOf(ToolItem item)
```

搜索工具栏，直至找到一个与参数匹配的按钮，并返回该按钮的索引值。

下面的代码说明如何创建一个新的 SWT 工具栏：

```
public class ToolbarClass {
    public static void main(String [] args) {
        Display display = new Display();
        final Shell shell = new Shell(display);
        shell.setSize(300, 200);
        shell.setText("ToolBar Example");

        ToolBar toolbar = new ToolBar(shell, SWT.NONE);
        toolbar.setBounds(0, 0, 200, 70);
        .
        .
        .
    }
}
```

下一步工作是在工具栏上添加一些按钮；详细内容请参阅下一节。

参考

9.4 节, 在工具栏上嵌入按钮; 9.5 节, 处理工具栏事件; 9.6 节, 在工具栏上嵌入组合框、文本小部件和菜单; *Eclipse* (O'Reilly) 一书的第 8 章。

9.4 在工具栏上嵌入按钮

问题

已经创建了一个工具栏, 你需要在其上添加一些按钮。

解决方案

要想把按钮放置到工具栏上, 可以使用 `ToolItem` 类, 并把要使用的 `ToolBar` 对象传递给 `ToolItem` 的构造函数。

讨论

下面是节选的一些 `ToolItem` 类的最常用的方法:

```
void addSelectionListener(SelectionListener listener)
```

将该监听程序添加到监听程序集合中, 当有工具栏按钮被选中时将通知该监听程序。

```
void setImage(Image image)
```

设置工具栏按钮的图像。

```
void setSelection(boolean selected)
```

设置工具栏按钮的选择状态。

```
void setText(String string)
```

设置工具栏按钮的文本。

```
void setToolTipText(String string)
```

设置工具栏按钮的工具提示文本。

```
void setWidth(int width)
```

设置工具栏按钮的宽度。

继续上一节的示例, 创建 5 个新的工具栏按钮, 并设置它们的标题。注意, 这里把 `ToolBar` 对象传递给 `ToolItem` 构造函数:

```
ToolItem toolItem1 = new ToolItem(toolbar, SWT.PUSH);
toolItem1.setText("Save");
ToolItem toolItem2 = new ToolItem(toolbar, SWT.PUSH);
toolItem2.setText("Save As");
ToolItem toolItem3 = new ToolItem(toolbar, SWT.PUSH);
toolItem3.setText("Print");
ToolItem toolItem4 = new ToolItem(toolbar, SWT.PUSH);
toolItem4.setText("Run");
ToolItem toolItem5 = new ToolItem(toolbar, SWT.PUSH);
toolItem5.setText("Help");
.
.
.
```

上述代码将按钮放置到工具栏上。要处理工具栏按钮的单击事件，请参阅下一节。

参考

9.3 节，创建工具栏；9.5 节，处理工具栏事件；9.6 节，在工具栏上嵌入组合框、文本小部件和菜单；*Eclipse* (O'Reilly) 一书的第 8 章。

9.5 处理工具栏事件

问题

你想捕获工具栏单击事件。

解决方案

在工具栏按钮中添加一个监听程序。只需新建一个 `Listener` 对象，并使用工具栏按钮的 `addListener` 方法添加该监听程序即可。

讨论

为了完成前两节中讨论的例子，新建一个 `Listener` 类的监听程序，用于处理按钮单击事件，并显示被单击按钮的标题（可以从工具按钮的 `getText` 方法获得），并将新建的监听程序添加到工具栏按钮中。通过例 9-2 可以了解如何使用监听程序。

例 9-2：创建工具栏

```
package org.cookbook.ch09;

import org.eclipse.swt.*;
import org.eclipse.swt.widgets.*;
```

```
public class ToolbarClass {

    public static void main(String [] args) {
        Display display = new Display();
        final Shell shell = new Shell(display);
        shell.setSize(300, 200);
        shell.setText("Toolbar Example");

        ToolBar toolbar = new ToolBar(shell, SWT.NONE);
        toolbar.setBounds(0, 0, 200, 70);

        ToolItem toolItem1 = new ToolItem(toolbar, SWT.PUSH);
        toolItem1.setText("Save");
        ToolItem toolItem2 = new ToolItem(toolbar, SWT.PUSH);
        toolItem2.setText("Save As");
        ToolItem toolItem3 = new ToolItem(toolbar, SWT.PUSH);
        toolItem3.setText("Print");
        ToolItem toolItem4 = new ToolItem(toolbar, SWT.PUSH);
        toolItem4.setText("Run");
        ToolItem toolItem5 = new ToolItem(toolbar, SWT.PUSH);
        toolItem5.setText("Help");

        final Text text = new Text(shell, SWT.BORDER);
        text.setBounds(55, 80, 200, 25);

        Listener toolbarListener = new Listener() {
            public void handleEvent(Event event) {
                ToolItem toolItem = (ToolItem)event.widget;
                String caption = toolItem.getText();
                text.setText("You clicked " + caption);
            }
        };

        toolItem1.addListener(SWT.Selection, toolbarListener);
        toolItem2.addListener(SWT.Selection, toolbarListener);
        toolItem3.addListener(SWT.Selection, toolbarListener);
        toolItem4.addListener(SWT.Selection, toolbarListener);
        toolItem5.addListener(SWT.Selection, toolbarListener);

        shell.open();

        while (!shell.isDisposed()) {
            if (!display.readAndDispatch())
                display.sleep();
        }
        display.dispose();
    }
}
```

运行结果如图 9-4 所示，工具栏出现在左上角（可以使用 `setBounds` 方法或 SWT 布局将其放置在任何需要的位置）。当用户单击一个按钮时，代码将报告用户的选择，如图 9-4 所示。

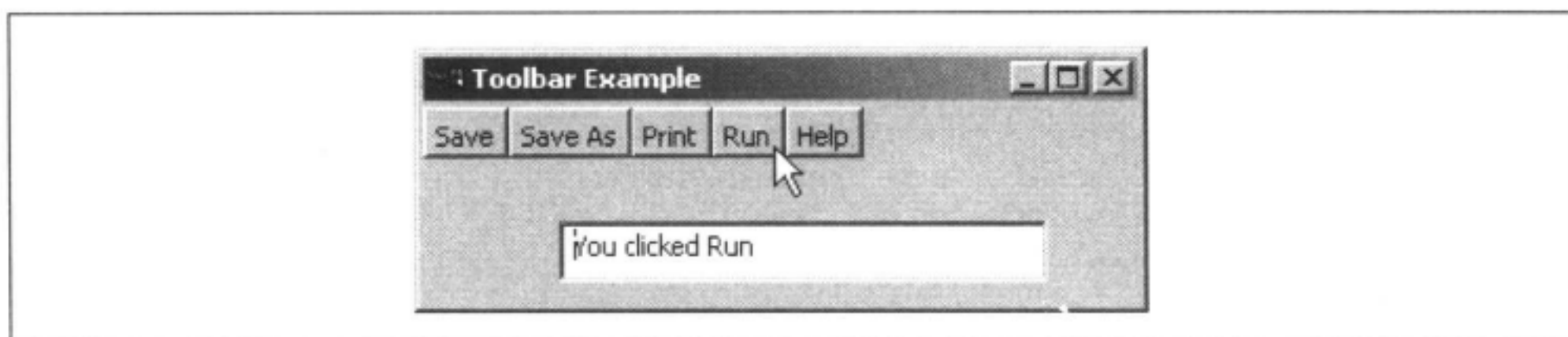


图 9-4：使用工具栏

参考

9.3 节，创建工具栏；9.4 节，在工具栏上嵌入按钮；9.6 节，在工具栏上嵌入组合框、文本小部件和菜单；*Eclipse* (O'Reilly) 一书的第 8 章。

9.6 在工具栏上嵌入组合框、文本小部件和菜单

问题

对于许多功能来讲，将按钮放在工具栏上是合适的。但如果你想给用户留下深刻的印象，可以添加其他的窗口小部件，如组合框或下拉菜单。

解决方案

对于组合框等窗口小部件而言，工具栏可以充当窗口小部件的容器。而且，可以从工具栏内部打开菜单。

讨论

在创建大多数窗口小部件时，可以指定要使用何种容器，而且可以通过使工具栏成为窗口小部件容器，在工具栏上添加窗口小部件。例如，下面是在工具栏上添加文本小部件的方法：

```
Text text = new Text(toolbar, SWT.BORDER);
```

而下面则是添加组合框小部件的方法：

```
Combo combo = new Combo(toolbar, SWT.READ_ONLY);
```

菜单的添加稍微复杂一些。在 SWT 中，用快捷菜单来支持工具栏菜单；可以用下面的代码设置快捷菜单在工具栏上的位置，并在必要时显示快捷菜单：

```
menu.setLocation(point.x, point.y);  
menu.setVisible(true);
```

参考

9.3 节，创建工具栏；9.4 节，在工具栏上嵌入按钮；9.5 节，处理工具栏事件；*Eclipse* (O'Reilly) 一书的第 8 章。

9.7 创建菜单系统

问题

你想在 SWT 应用程序中创建并添加一个菜单系统。

解决方案

使用 `Menu` 和 `MenuItem` 对象创建菜单系统。新建一个 `Menu` 对象，用于菜单系统本身，新建若干 `MenuItem` 对象，用于各个菜单及菜单项。

讨论

使用 `Menu` 和 `MenuItem` 对象可以在 SWT 应用程序中创建菜单系统。为了掌握这一过程，在此我们打算创建一个菜单系统，其中具有 `File` 和 `Edit` 两个菜单。当用户选择一个菜单项时，将在一个文本小部件中显示他所选择的菜单项。

这个例子就是本书站点的 *MenuApp* 项目。首先，使用与菜单栏对应的 `Menu` 对象创建一个标准的菜单系统。下面是节选的一些 `Menu` 类的常用方法：

```
void addMenuListener(MenuListener listener)
```

将该监听程序添加到监听程序集合中，当有菜单被隐藏或显示时将通知该监听程序。

```
MenuItem getItem(int index)
```

返回给定索引的菜单项。

```
int getItemCount()
```

返回菜单中包含的菜单项的数量。

```
void setVisible(boolean visible)
```

如果参数为真，显示菜单；否则，隐藏菜单。

下面是我们的菜单栏，它是用 SWT.BAR 样式创建的：

```
public class MenuClass
{
    Display display;
    Shell shell;
    Menu menuBar;
    Text text;

    public MenuClass()
    {
        display = new Display();
        shell = new Shell(display);
        shell.setText("Menu Example");
        shell.setSize(300, 200);

        text = new Text(shell, SWT.BORDER);
        text.setBounds(80, 50, 150, 25);

        menuBar = new Menu(shell, SWT.BAR);
        .
        .
        .
    }
}
```

在创建了与菜单栏对应的对象之后，创建菜单中的菜单项，如 File 菜单和 Edit 菜单。然而，不是使用 Menu 类，而是使用 MenuItem 类。下面是节选的一些 MenuItem 类的常用方法：

```
void addSelectionListener(SelectionListener listener)
```

将该监听程序添加到监听程序集合中，当有菜单项被选中时将通知该监听程序。

```
boolean getSelection()
```

如果菜单项被选中，返回 true；否则，返回 false。

```
boolean isEnabled()
```

如果菜单项被启用，返回 true；否则，返回 false。

```
void setAccelerator(int accelerator)
```

设置菜单项的加速键。

```
void setEnabled(boolean enabled)
```

如果参数为真，启用菜单项；否则，禁用菜单项。

```
void setImage(Image image)
```

根据参数设置菜单项将显示的图像。

```
void setMenu(Menu menu)
```

根据参数设置菜单项的下拉菜单。

```
void setText(String string)
```

设置菜单项的文本。

下面说明如何创建 File 菜单头, 并将其标题设置为 “File”, 助记符设置为 “F” (这意味着, 在 Windows 中按 Alt-F 键, 在 Mac OS X 按 Apple-F 键, 也可以打开此菜单):

```
public class MenuClass
{
    Display display;
    Shell shell;
    Menu menuBar;
    MenuItem fileMenuHeader;
    Text text;

    public MenuClass()
    {
        display = new Display();
        shell = new Shell(display);
        shell.setText("Menu Example");
        shell.setSize(300, 200);

        text = new Text(shell, SWT.BORDER);
        text.setBounds(80, 50, 150, 25);

        menuBar = new Menu(shell, SWT.BAR);
        fileMenuHeader = new MenuItem(menuBar, SWT.CASCADE);
        fileMenuHeader.setText("&File");
        .
        .
        .
    }
}
```

要创建实际的 File 下拉菜单, 可使用 SWT.DROP_DOWN 样式的 Menu 类, 并把新菜单添加到 File 菜单头中, 如下所示:

```
public class MenuClass
{
    Display display;
    Shell shell;
    Menu menuBar, fileMenu;
    MenuItem fileMenuHeader;
    Text text;

    public MenuClass()
    {
        .
        .
        .
    }
}
```

```
menuBar = new Menu(shell, SWT.BAR);  
fileMenuHeader = new MenuItem(menuBar, SWT.CASCADE);  
fileMenuHeader.setText("&File");  
  
fileMenu = new Menu(shell, SWT.DROP_DOWN);  
fileMenuHeader.setMenu(fileMenu);  
.  
.  
.
```

这将生成一个File菜单，如图9-5所示。如何在此菜单中添加菜单项呢？请阅读下一节。

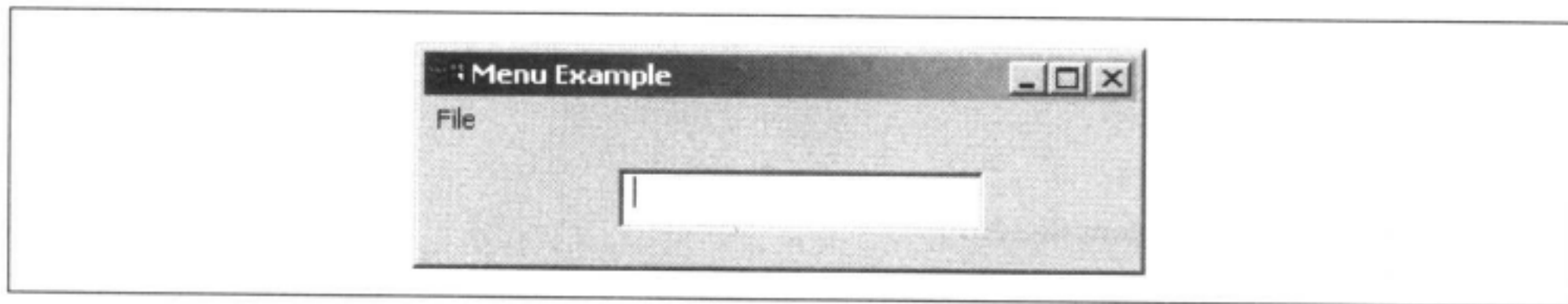


图 9-5: File 菜单

参考

9.8 节，创建文本菜单项；9.9 节，创建图像菜单项；9.10 节，创建单选菜单项；9.11 节，创建菜单项加速键和助记符；9.12 节，启用和禁用菜单项；*Eclipse* (O'Reilly) 一书的第 8 章。

9.8 创建文本菜单项

问题

你想添加能够显示菜单的文本标题，并能够处理这些菜单项的选择事件的菜单项。

解决方案

使用 `MenuItem` 类和 `SelectionAdapter` 类。创建一个 `MenuItem` 对象，使用 `addSelectionListener` 方法在其中添加一个监听程序，并用 `setText` 方法设置菜单项的文本标题。

讨论

要在上一节开始的菜单示例中添加一个 `File` → `Save` 菜单项，可新建一个标题为 `Save`

的MenuItem对象, 把fileMenu对象传递给其构造函数, 并将该菜单项的样式设置为SWT.PUSH:

```
public MenuClass()  
{  
    .  
    .  
    .  
    menuBar = new Menu(shell, SWT.BAR);  
    fileMenuHeader = new MenuItem(menuBar, SWT.CASCADE);  
    fileMenuHeader.setText("&File");  
  
    fileMenu = new Menu(shell, SWT.DROP_DOWN);  
    fileMenuHeader.setMenu(fileMenu);  
  
    fileSaveItem = new MenuItem(fileMenu, SWT.PUSH);  
    fileSaveItem.setText("&Save");  
    .  
    .  
    .  
}
```

为了激活这个菜单项, 将它连接到一个选择监听程序类, 即MenuListener类:

```
fileSaveItem.addSelectionListener(new MenuItemListener());
```

MenuListener类将扩展SelectionAdapter类。如第8章所述, 所有SWT监听程序接口都有适配器类, 其中包括接口的所有方法的占位实现; 如果想扩展一个适配器类, 必须仅实现要覆盖的方法。在本例中, 将通过检索引起菜单项选择事件的菜单项小部件, 来显示选定菜单项的文本。注意, 如果所选菜单项为File → Exit, 将退出程序:

```
class MenuItemListener extends SelectionAdapter  
{  
    public void widgetSelected(SelectionEvent event)  
    {  
        if(((MenuItem) event.widget).getText().equals("E&xit")){  
            shell.close();  
        }  
        text.setText("You selected " + ((MenuItem) event.widget).getText());  
    }  
}
```

这样就完成了File → Save菜单项的代码; 在这个例子中, 我们还将添加File → Exit菜单项和Edit → Copy菜单项, 如例9-3所示。

例9-3: SWT 菜单

```
package org.cookbook.ch09;  
  
import org.eclipse.swt.widgets.*;  
import org.eclipse.swt.SWT;  
import org.eclipse.swt.events.*;
```

```
public class MenuClass
{
    Display display;
    Shell shell;
    Menu menuBar, fileMenu, editMenu;
    MenuItem fileMenuHeader, editMenuHeader;
    MenuItem fileExitItem, fileSaveItem, editCopyItem;
    Text text;

    public MenuClass()
    {
        display = new Display();
        shell = new Shell(display);
        shell.setText("Menu Example");
        shell.setSize(300, 200);

        text = new Text(shell, SWT.BORDER);
        text.setBounds(80, 50, 150, 25);

        menuBar = new Menu(shell, SWT.BAR);
        fileMenuHeader = new MenuItem(menuBar, SWT.CASCADE);
        fileMenuHeader.setText("&File");

        fileMenu = new Menu(shell, SWT.DROP_DOWN);
        fileMenuHeader.setMenu(fileMenu);

        fileSaveItem = new MenuItem(fileMenu, SWT.PUSH);
        fileSaveItem.setText("&Save");

        fileExitItem = new MenuItem(fileMenu, SWT.PUSH);
        fileExitItem.setText("E&xit");

        editMenuHeader = new MenuItem(menuBar, SWT.CASCADE);
        editMenuHeader.setText("&Edit");

        editMenu = new Menu(shell, SWT.DROP_DOWN);
        editMenuHeader.setMenu(editMenu);

        editCopyItem = new MenuItem(editMenu, SWT.PUSH);
        editCopyItem.setText("&Copy");

        fileExitItem.addSelectionListener(new MenuItemListener());
        fileSaveItem.addSelectionListener(new MenuItemListener());
        editCopyItem.addSelectionListener(new MenuItemListener());

        shell.setMenuBar(menuBar);
        shell.open();
        while (!shell.isDisposed())
        {
            if (!display.readAndDispatch())
                display.sleep();
        }
        display.dispose();
    }
}
```

```
class MenuItemListener extends SelectionAdapter
{
    public void widgetSelected(SelectionEvent event)
    {
        if(((MenuItem) event.widget).getText().equals("E&xit")){
            shell.close();
        }
        text.setText("You selected " + ((MenuItem) event.widget).getText());
    }
}

public static void main(String[] args)
{
    MenuClass menuExample = new MenuClass();
}
```

运行结果如图 9-6 所示，其中我们正在选择 File → Save 菜单项。

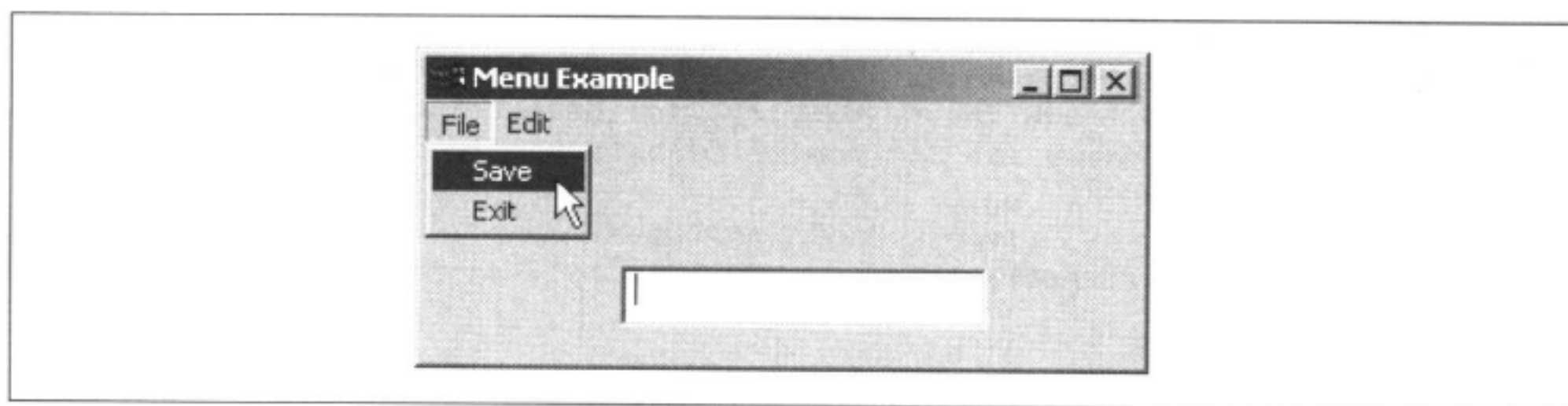


图 9-6: 打开 File 菜单

如果选择 File → Save，可以看到所选菜单项显示在应用程序的文小部件中，如图 9-7 所示。

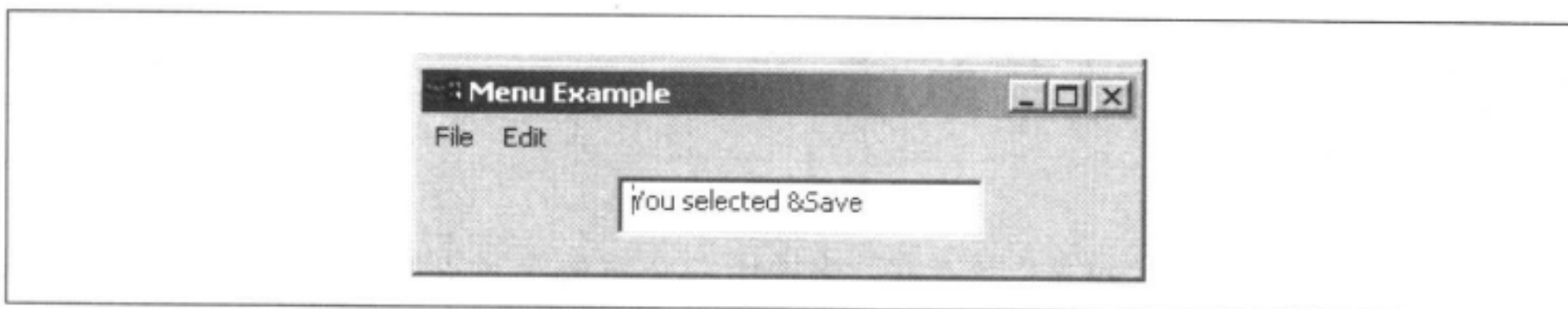


图 9-7: 选择一个菜单项

参考

9.7 节，创建菜单系统；9.9 节，创建图像菜单项；9.10 节，创建单选菜单项；9.11 节，创建菜单项加速键和助记符；9.12 节，启用和禁用菜单项；*Eclipse* (O'Reilly) 一书的第 8 章。

9.9 创建图像菜单项

问题

你想在菜单项中显示图像。

解决方案

使用 MenuItem 类的 setImage 方法。

讨论

可以给 MenuItem 对象的 setImage 方法传递一个 Image 对象, 以便为菜单项添加图像。

下面是一个例子:

```
Display display = new Display ();
final Image image = new Image (display, "c:\\helpitem.jpg");
.
.
.
menuItem.setImage(image);
```

参考

9.7 节, 创建菜单系统; 9.8 节, 创建文本菜单项; 9.10 节, 创建单选菜单项; 9.11 节, 创建菜单项加速键和助记符; 9.12 节, 启用和禁用菜单项; *Eclipse* (O'Reilly) 一书的第 8 章。

9.10 创建单选菜单项

问题

你想在一个菜单中添加“可选”菜单项, 当选中一个可选菜单项时, 它将保持选中状态, 直到有另一个可选菜单项被选中为止。

解决方案

使用 SWT 单选菜单项。通过将 MenuItem 对象的样式设置为 SWT.RADIO, 可以创建单选菜单项。

讨论

例如, 假设你想在本章前面开发的菜单示例中的 File 菜单中添加两个单选菜单项, 用这两个单选菜单项把应用程序中使用的语言设置为德语或英语。可以像下面这样创建两个新的 SWT.RADIO 菜单项:

```
fileEnglishItem = new MenuItem(fileMenu, SWT.RADIO);
fileEnglishItem.setText("English");

fileGermanItem = new MenuItem(fileMenu, SWT.RADIO);
fileGermanItem.setText("German");
```

为了处理菜单选择事件, 还应创建一个名为 `RadioItemListener` 的类, 此类扩展了 `SelectionAdapter` 类。在下面的代码中, 我们将捕获用户选择的单选菜单项, 并报告所使用的语言 (如果只想查明是否有一个菜单项的单项按钮被选择, 可调用它的 `getSelection` 方法):

```
class RadioItemListener extends SelectionAdapter
{
    public void widgetSelected(SelectionEvent event)
    {
        MenuItem item = (MenuItem)event.widget;
        text.setText(item.getText() + " is on.");
    }
}
```

运行结果如图 9-8 所示, 从中可以看到 German 和 English 单选菜单项。当选择其中一个菜单项时, SWT 将关闭或打开该菜单项。

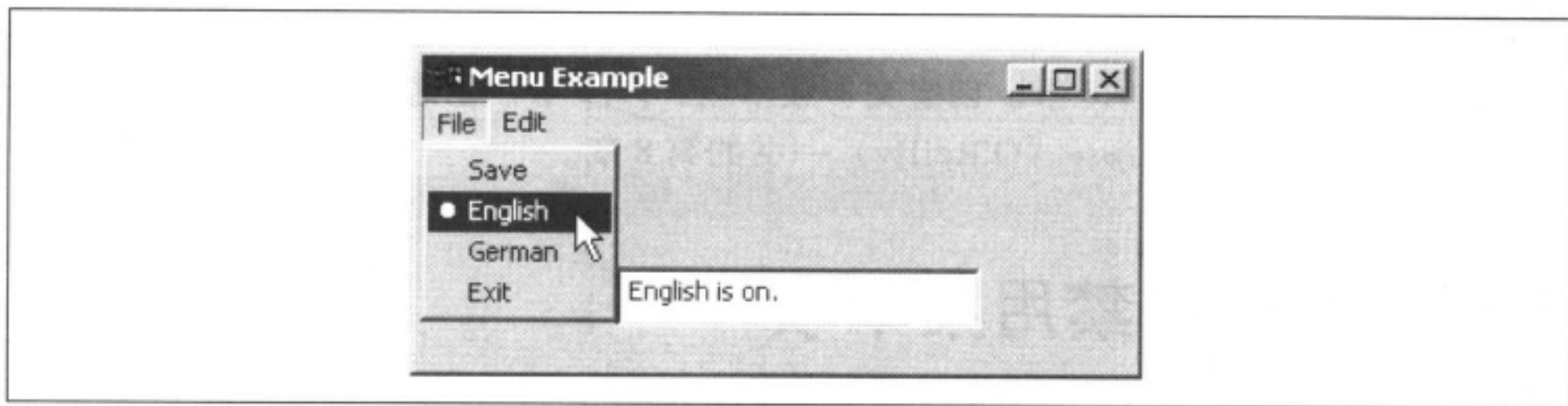


图 9-8: 单选菜单项

参考

9.7 节, 创建菜单系统; 9.8 节, 创建文本菜单项; 9.9 节, 创建图像菜单项; 9.11 节, 创建菜单项加速键和助记符; 9.12 节, 启用和禁用菜单项。

9.11 创建菜单项加速键和助记符

问题

你需要为菜单项添加键盘访问方式。

解决方案

SWT支持菜单项加速键和助记符。要创建加速键，可使用菜单项的 `setAccelerator` 方法；要创建助记符，可在菜单项的标题中插入一个“与”符号（&），将其放在要用作助记符的那个字符之前（要确保在同时显示的助记符中，它是唯一的）。

讨论

助记符是可以作为组合键的一部分而访问的键，用于选择一个菜单项。例如，如果一个菜单项的助记符是 S，通过在 Windows 中按 Alt-S 组合键，在 Mac OS X 中按 Apple-S 组合键，可以访问该菜单项。只有在相应的菜单已经打开时，才可以使用助记符。

而另一方面，无论相应的菜单是否已经打开，随时都可以使用加速键。作为一个例子，下面的代码说明了如何将一个菜单项的助记符设置为 S，而加速键设置为 Ctrl-S：

```
menuItem.setText("&Spell Check\tCtrl+S");  
menuItem.setAccelerator (SWT.CTRL + 'S');
```

参考

9.7 节，创建菜单系统；9.8 节，创建文本菜单项；9.10 节，创建单选菜单项；9.12 节，启用和禁用菜单项；*Eclipse* (O'Reilly) 一书的第 8 章。

9.12 启用和禁用菜单项

问题

你想禁用不适合选择的菜单项。

解决方案

要启用和禁用菜单项，可使用 `MenuItem` 类的 `setEnabled` 方法。

讨论

为 `setEnabled` 传递一个布尔值；传递 `true` 时启用菜单项，传递 `false` 时禁用菜单项。

参考

9.7 节，创建菜单系统；9.8 节，创建文本菜单项；9.9 节，创建图像菜单项；9.10 节，创建单选菜单项；9.11 节，创建菜单项加速键和助记符。

9.13 创建菜单分隔符

问题

你想使用水平线将菜单项分成几组。

解决方案

新建一个菜单项，通过将其样式设置为 `SWT.SEPARATOR`，使其充当分隔符。

讨论

下面的示例代码创建了一个菜单分隔符，并将其添加到 `File` 菜单中：

```
fileMenuSeparator = new MenuItem(fileMenu, SWT.SEPARATOR);
```

参考

9.7 节，创建菜单系统；9.8 节，创建文本菜单项；9.9 节，创建图像菜单项；9.10 节，创建单选菜单项；9.11 节，创建菜单项加速键和助记符；*Eclipse* (O'Reilly) 一书的第 8 章。

9.14 创建表格

问题

你有大量的数据需要进行直观展示，并想分栏排列这些数据。

解决方案

使用基于 `Table` 类的 SWT 表格。SWT 表格可以以列的形式显示文本、图像、复选框等等。

讨论

下面是节选的一些 Table 类的方法：

```
void addSelectionListener(SelectionListener listener)
```

将该监听程序添加到监听程序集合中，当表格的选择改变时将通知该监听程序。

```
void deselect(int index)
```

在表格中给定零相对索引处取消对表格项的选择。

```
TableItem[] getSelection()
```

返回一个由表格中被选中 TableItem 对象构成的数组。

```
int getSelectionIndex()
```

返回表格中当前选中的那个表格项的零相对索引（如果没有选中任何表格项，则返回 -1）。

```
int[] getSelectionIndices()
```

返回表格中当前选中的那些表格项的零相对索引。

```
boolean isSelected(int index)
```

如果表格项被选中，返回 true，否则返回 false。

```
void select(int index)
```

选择表格中给定零相对索引处的表格项。

作为一个例子（本书站点的 *TableApp* 项目），我们创建一个简单的表格，用于显示捕获选择事件的文本项。创建一个新的表格，并使用 TableItem 类在表格中填充一些表格项，然后在一个文本小部件中显示选择的表格项。下面是 TableItem 类的一些常用的方法：

```
boolean getChecked()
```

如果表格项的复选框被选中，返回 true，否则返回 false。

```
boolean getGrayed()
```

如果表格项呈灰色，返回 true，否则返回 false。

```
void setChecked(boolean checked)
```

设置该表格项的复选框的选中状态。

```
void setGrayed(boolean grayed)
```

设置该表格项的复选框的灰色状态。

```
void setImage(Image image)
```

设置表格项的图像。

```
void setText(String string)
```

设置表格项的文本。

下面这个例子说明了如何创建表格:

```
Table table = new Table(shell, SWT.BORDER | SWT.V_SCROLL | SWT.H_SCROLL);
```

而下面的代码说明了如何用 `TableItem` 对象填充表格:

```
for (int loopIndex=0; loopIndex < 24; loopIndex++) {  
    TableItem item = new TableItem (table, SWT.NULL);  
    item.setText("Item " + loopIndex);  
}
```

剩下的工作就是处理表格项选择事件,这可以用例9-4所示的代码来完成;可以使用传递给 `handleEvent` 方法的事件对象的表格项成员,重新获得选择的表格项。

例 9-4: SWT 表格

```
package org.cookbook.ch09;  
  
import org.eclipse.swt.*;  
import org.eclipse.swt.widgets.*;  
  
public class TableClass  
{  
    public static void main(String[] args)  
    {  
        Display display = new Display();  
        Shell shell = new Shell(display);  
        shell.setSize(260, 300);  
        shell.setText("Table Example");  
  
        final Text text = new Text(shell, SWT.BORDER);  
        text.setBounds(25, 240, 200, 25);  
  
        Table table = new Table(shell, SWT.BORDER | SWT.V_SCROLL | SWT.H_SCROLL);  
  
        for (int loopIndex=0; loopIndex < 24; loopIndex++) {  
            TableItem item = new TableItem (table, SWT.NULL);  
            item.setText("Item " + loopIndex);  
        }  
  
        table.setBounds(25, 25, 200, 200);  
  
        table.addListener(SWT.Selection, new Listener()  
        {  
            public void handleEvent(Event event)  
            {  
                text.setText("You selected " + event.item);  
            }  
        });  
    }  
}
```

```
shell.open();
while (!shell.isDisposed())
{
    if (!display.readAndDispatch())
        display.sleep();
}
display.dispose();
}
```

运行结果如图 9-9 所示。当选择表格中的某一项时，应用程序将指示选择了哪一项。

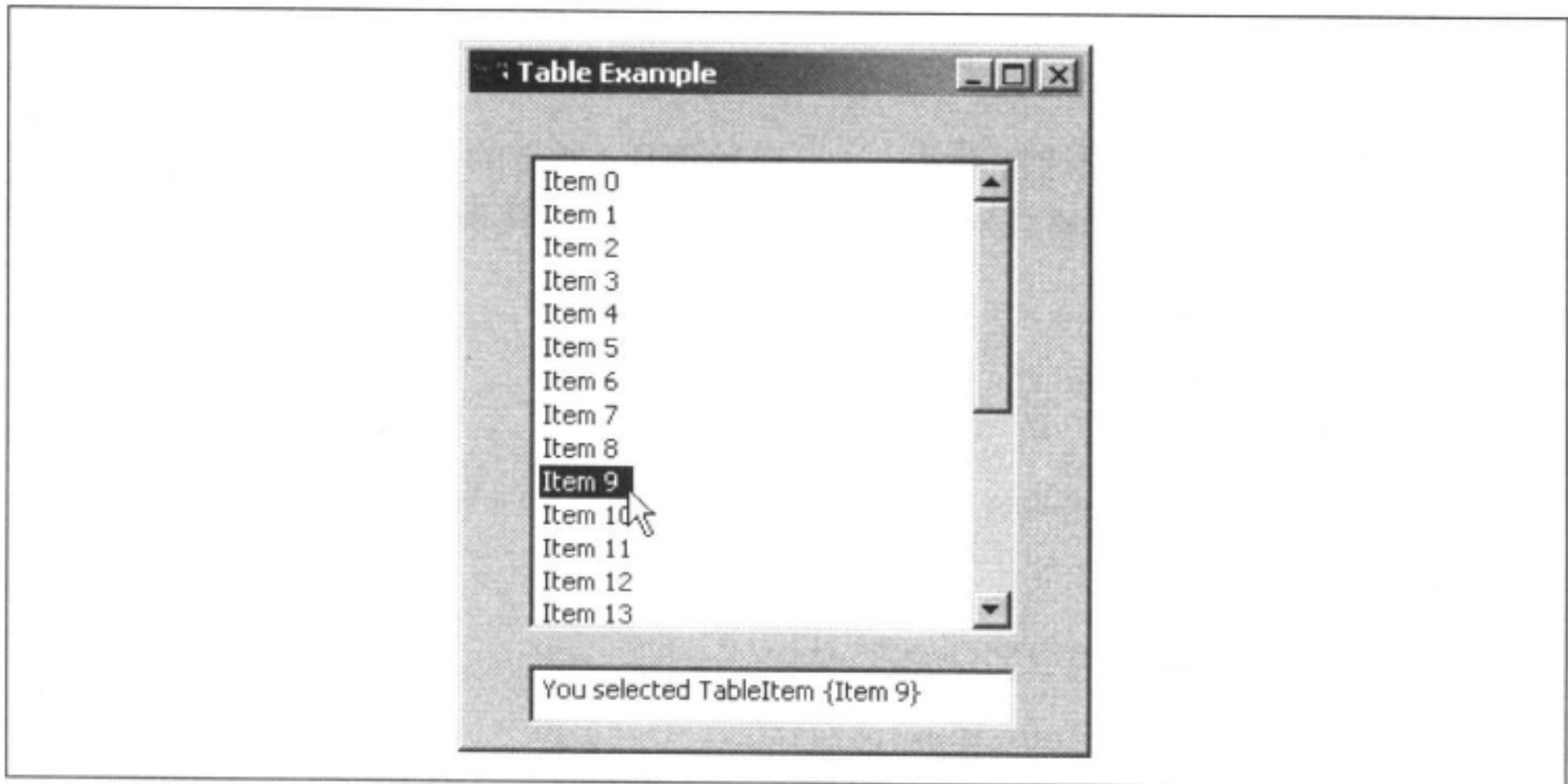


图 9-9：一个简单的表格

到目前为止，这个例子看起来还不错，但这个基本的例子仅仅是一个开始（事实上，这个简单的表格看起来非常像一个简单的列表小部件）。要添加表格列、复选标记、图像等，请参阅下面几节。

注意：默认情况下，表格只允许进行单项选择。为了能够进行多项选择，可使用 `SWT.MULTI` 样式而非 `SWT.SINGLE` 样式，来创建表格。

Eclipse 3.0

在 Eclipse 3.0 中，SWT 表格小部件支持设置单元格的前景色和背景色。另外，通过 `Table` 小部件可以设置行或者单元格的字体。

参考

9.15 节, 创建表格列; 9.16 节, 为表格项添加复选标记; 9.17 节, 启用和禁用表格项; 9.18 节, 为表格项添加图像。

9.15 创建表格列

问题

你需要在 SWT 表格小部件中添加几个列。

解决方案

为每一列创建一个 TableColumn 对象, 并把要使用的 Table 对象传递给 TableColumn 对象的构造函数。然后创建 TableItem 对象, 并分别设置每一列的文本。

讨论

要在 SWT 表格中添加列, 首先应在表格中打开表格头:

```
Table table = new Table(shell, SWT.BORDER | SWT.V_SCROLL | SWT.H_SCROLL);
table.setHeaderVisible (true);
```

然后使用 TableColumn 对象创建列。本例中将添加 4 列:

```
String[] titles = {"Col 1", "Col 2", "Col 3", "Col 4"};

for (int loopIndex = 0; loopIndex < titles.length; loopIndex++) {
    TableColumn column = new TableColumn(table, SWT.NULL);
    column.setText(titles[loopIndex]);
}
```

在表格中填充表格项时, 还可以通过传递列号以及要使用的文本, 为每一列设置文本:

```
for (int loopIndex=0; loopIndex < 24; loopIndex++) {
    TableItem item = new TableItem (table, SWT.NULL);
    item.setText(0, "Item " + loopIndex);
    item.setText(1, "Yes");
    item.setText(2, "No");
    item.setText(3, "A table item");
}
```

还可以压缩列, 以便根据其中包含的文本的宽度调整列宽:

```
for (int loopIndex = 0; loopIndex < titles.length; loopIndex++) {
    table.getColumn(loopIndex).pack();
}
```

就这么简单。结果如图9-10所示。当选择一个表格项时，应用程序将显示所选的表格项。

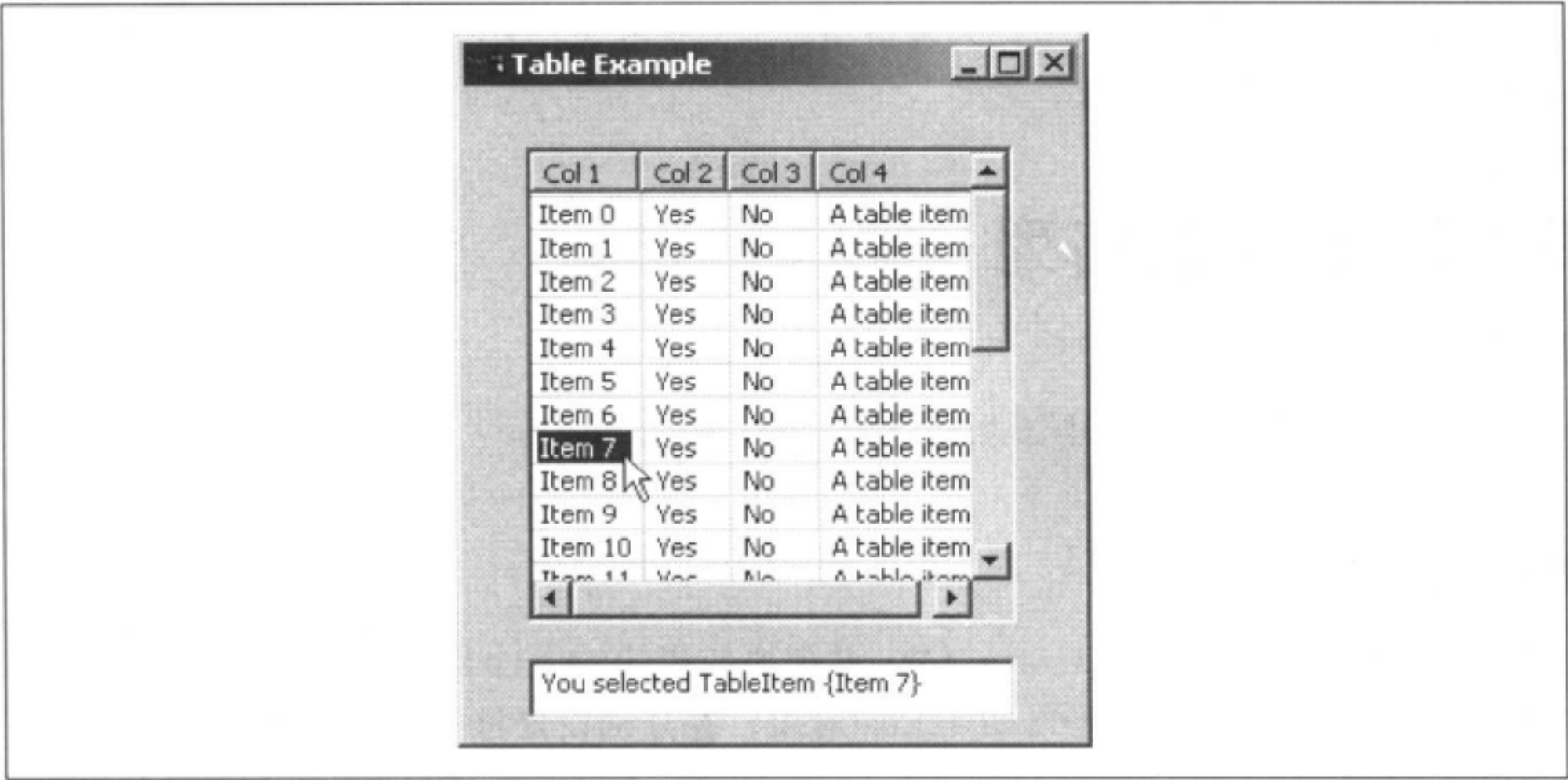


图 9-10：一个多列表格

Eclipse 3.0

在 Eclipse 3.0 中，通过调用 `Table.showColumn(TableColumn)` 方法可以滚动列，使其显示在视图中。

参考

9.14 节，创建表格；9.16 节，为表格项添加复选标记；9.17 节，启用和禁用表格项；9.18 节，为表格项添加图像。

9.16 为表格项添加复选标记

问题

你想使用户能够选择 SWT 表格小部件中的各个表格项。

解决方案

在创建表格时，采用 `SWT.CHECK` 样式。使用 `getChecked` 方法可以判断表格项是否被选中，如果被选中，则返回 `true`。

讨论

在表格中添加了复选标记之后,可以使用表格的 `getSelection` 方法判断哪些表格项被选中了,这个方法将返回一个又选定的 `TableItem` 对象构成的数组;也可以使用 `getSelectionIndex` 方法,该方法将返回当前选定表格项的索引;或使用 `getSelectionIndices` 方法,该方法返回一个由多选表格中选定表格项的索引构成的 `int` 型数组。要查看一个表格项是否被选中,可调用它的 `getChecked` 方法,或者使用 `setChecked` 方法明确地选中它。

例如,下面的代码说明了如何创建带有复选框的表格:

```
Table table = new Table(shell, SWT.CHECK | SWT.BORDER | SWT.V_SCROLL | SWT.H_SCROLL);
```

通过查看 `handleEvent` 方法中传递的 `Event` 对象的 `detail` 成员,可以确定最近单击的表格项是否已被选中:

```
table.addListener(SWT.Selection, new Listener()
{
    public void handleEvent(Event event)
    {
        if(event.detail == SWT.CHECK){
            text.setText("You checked " + event.item);
        } else {
            text.setText("You selected " + event.item);
        }
    }
});
```

结果如图 9-11 所示。如图所示,应用程序显示了已经选中的表格项。

参考

9.15 节, 创建表格列; 9.16 节, 为表格项添加复选标记; 9.17 节, 启用和禁用表格项; 9.18 节, 为表格项添加图像。

9.17 启用和禁用表格项

问题

你想把表格项变灰,使之不可用。

解决方案

使用表格项的 `setGrayed` 方法。

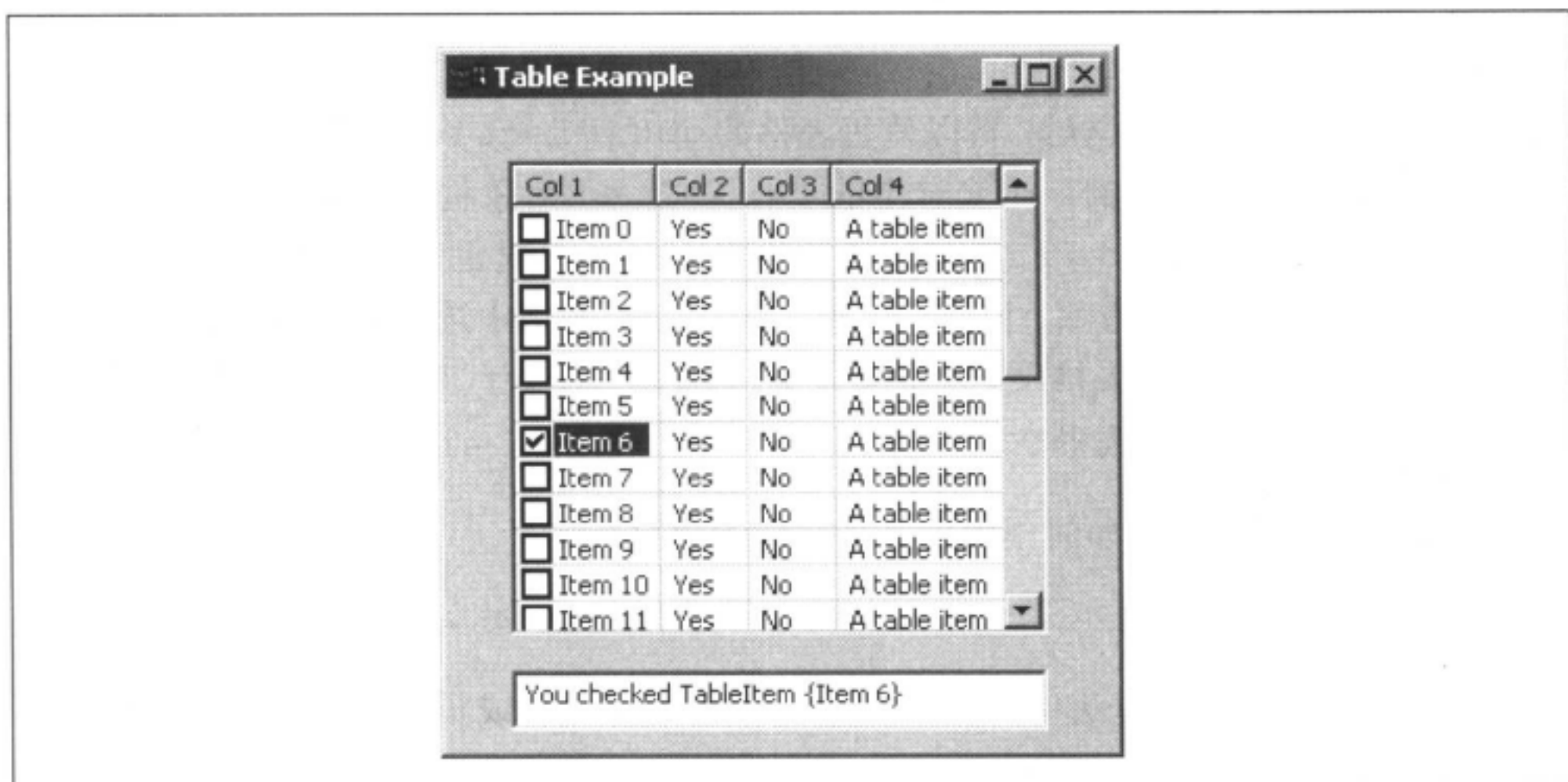


图 9-11：具有复选标记的表格

讨论

要禁用一个表格项，可用一个 `true` 值来调用该表格项的 `setGrayed` 方法；要再次启用该表格项，可为 `setGrayed` 方法传递一个 `false` 值。

9.18 为表格项添加图像

问题

你想为表格项添加图像。

解决方案

要为一个表格项添加图像，可调用该表格项的 `setImage` 方法。

讨论

只需将一个 `Image` 对象传递给表格项的 `setImage` 方法，即可为该表格项添加一个图像。关于创建 `Image` 对象的详细介绍，请参阅本章前面的 9.9 节。

参考

9.9 节, 创建图像菜单项; 9.14 节, 创建表格; 9.16 节, 为表格项添加复选标记; 9.17 节, 启用和禁用表格项。

9.19 在 SWT 内部使用 Swing 和 AWT

问题

你想在一个 SWT 应用程序内部使用 Swing 或 AWT 图形元素。

解决方案

在 Eclipse 3.0 中, SWT 支持将 Swing/AWT 窗口小部件嵌入 SWT 窗口小部件。在 3.0 版之前, 这种支持仅适用于 Windows。在 Eclipse 3.0 中, 现在这种支持不仅适用于 Windows 下的 JDK 1.4 及其更新版本, 而且适用于 GTK 和 Motif, 并可以方便地访问 JDK 1.5。

讨论

要在 SWT 中使用 AWT 和 Swing 元素, 需要使用 SWT_AWT 类。作为一个例子 (本书站点的 *SwingAWTApp* 项目), 我们将创建一个应用程序, 其中使用了 AWT 框架和面板以及 Swing 按钮和文本控件。首先, 新建一个 SWT 复合小部件:

```
Composite composite = new Composite(shell, SWT.EMBEDDED);
composite.setBounds(20, 20, 300, 200);
composite.setLayout(new RowLayout());
```

然后, 使用 SWT_AWT.new_Frame 方法将一个 AWT Frame 窗口对象添加到复合小部件中, 并在框架中添加一个 AWT Panel 对象:

```
java.awt.Frame frame = SWT_AWT.new_Frame(composite);
java.awt.Panel panel = new java.awt.Panel();
frame.add(panel);
```

现在, 可以使用 Swing 控件了。在这个例子中, 我们将在 AWT 面板上添加一个 Swing 按钮和一个 Swing 文本控件:

```
final javax.swing.JButton button = new javax.swing.JButton("Click Me");
final javax.swing.JTextField text = new javax.swing.JTextField(20);
panel.add(button);
panel.add(text);
```


余下的工作就是把该按钮连接到一个监听程序,以便在按钮被单击时显示一条消息。通过例9-5可以了解这是如何完成的。

例9-5: 在 SWT 中使用 Swing 和 AWT

```
package org.cookbook.ch09;

import java.awt.event.*;

import org.eclipse.swt.*;
import org.eclipse.swt.widgets.*;
import org.eclipse.swt.layout.*;
import org.eclipse.swt.awt.SWT_AWT;

public class SwingAWTClass {

    public static void main(String[] args) {
        final Display display = new Display();
        final Shell shell = new Shell(display);
        shell.setText("Using Swing and AWT");
        shell.setSize(350, 280);

        Composite composite = new Composite(shell, SWT.EMBEDDED);
        composite.setBounds(20, 20, 300, 200);
        composite.setLayout(new RowLayout());

        java.awt.Frame frame = SWT_AWT.new_Frame(composite);
        java.awt.Panel panel = new java.awt.Panel();
        frame.add(panel);

        final javax.swing.JButton button = new javax.swing.JButton("Click Me");
        final javax.swing.JTextField text = new javax.swing.JTextField(20);

        panel.add(button);
        panel.add(text);

        button.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent event) {
                text.setText("Yep, it works.");
            }
        });

        shell.open();
        while(!shell.isDisposed()) {
            if (!display.readAndDispatch()) display.sleep();
        }
        display.dispose();
    }
}
```

运行这个应用程序,得到如图9-12所示的结果,在SWT应用程序中的AWT框架中,有一个可用的Swing按钮和文本控件。

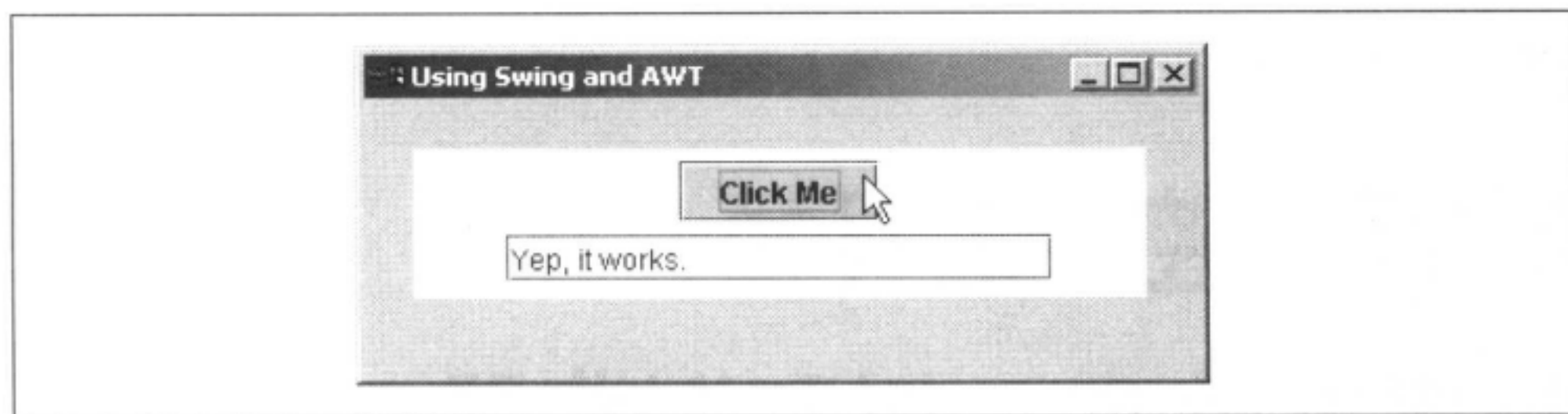


图 9-12: 混合使用 AWT、Swing 和 SWT

第 10 章

SWT: 控件工具栏、 标签文件夹、树和浏览器

10.0 简介

这是关于 SWT 的最后一章，介绍一些更高级的窗口小部件：控件工具栏 (coolbar)、标签文件夹 (tab folder)、树和浏览器。所有这些窗口小部件都有各自的用途，本章将考察这些窗口小部件的功能。

浏览器小部件是 Eclipse 3.0 中新增的小部件；在以前的版本中，仅在 Windows 中支持浏览器，而且必须使用 OLE 技术进行启动。但在 Eclipse 3.0 中，浏览器小部件内置到了 SWT 中，而且随着时间的推移，越来越多的操作系统支持浏览器小部件。

10.1 创建 SWT 标签文件夹

问题

在应用程序中，屏幕空间是一种宝贵资源，而你想把窗口小部件分成一组页。

解决方案

尝试使用一个标签文件夹小部件，使用这种小部件可以堆叠页式小部件。新建一个 TabFolder 对象，在其中添加几个 TabItem 对象，并使用 TabItem 对象作为窗口小部件容器。

讨论

标签文件夹非常适合于创建和堆叠窗口小部件。只需像本书代码中的*TabApp*示例一样，创建一个*TabFolder*类的对象即可。下面是节选的一些最常用的*TabFolder*类的方法：

```
void addSelectionListener(SelectionListener listener)
```

将该监听程序添加到监听程序集合中，当标签文件夹的选择变化时将通知该监听程序。

```
TabItem getItem(int index)
```

返回标签文件夹上给定零相对索引处的项目。

```
int getItemCount()
```

返回标签文件夹上项目的数量。

```
TabItem[].getItems()
```

返回一个*TabItem*对象数组，这些对象是标签文件夹上的项目。

```
void setSelection(int index)
```

选择标签文件夹上给定零相对索引处的项目。

下面的代码说明*TabApp*应用程序如何创建标签文件夹：

```
final TabFolder tabFolder = new TabFolder(shell, SWT.BORDER);  
.  
.  
.
```

在创建了一个标签文件夹之后，可以根据需要在标签文件夹中添加许多*TabItem*对象，以创建标签页。下面是节选的一些最有用的*TabItem*方法：

```
TabFolder getParent()
```

返回标签项的父对象，它必须是一个*TabFolder*对象。

```
void setControl(Control control)
```

设置用于填充标签项的客户区的控件。

```
void setImage(Image image)
```

设置标签项的图像。

```
void setText(String string)
```

设置标签项的文本。

```
void setToolTipText(String string)
```

设置标签项的工具提示文本。

在这个例子中，我们将新建 10 个 `TabItem` 对象，并通过把 `tabFolder` 对象传递给 `TabItem` 构造函数将这些对象连接到标签文件夹上，从而在标签文件夹上添加 10 个标签页：

```
final TabFolder tabFolder = new TabFolder(shell, SWT.BORDER);

for (int loopIndex = 0; loopIndex < 10; loopIndex++)
{
    TabItem tabItem = new TabItem(tabFolder, SWT.NULL);
    tabItem.setText("Tab " + loopIndex);
    .
    .
    .
}
```

余下的工作就是使用标签页的 `setControl` 方法，在每个标签页上添加窗口小部件，即本例中要使用的文本小部件，并打开 `shell`，如例 10-1 所示。

例 10-1：使用 SWT 标签文件夹

```
package org.cookbook.ch10;

import org.eclipse.swt.*;
import org.eclipse.swt.widgets.*;

public class TabClass
{
    public static void main(String[] args)
    {
        Display display = new Display();
        final Shell shell = new Shell(display);
        shell.setText("Tab Folder Example");
        shell.setSize(450, 250);

        final TabFolder tabFolder = new TabFolder(shell, SWT.BORDER);

        for (int loopIndex = 0; loopIndex < 10; loopIndex++)
        {
            TabItem tabItem = new TabItem(tabFolder, SWT.NULL);
            tabItem.setText("Tab " + loopIndex);

            Text text = new Text(tabFolder, SWT.BORDER);
            text.setText("This is page " + loopIndex);
            tabItem.setControl(text);
        }
        tabFolder.setSize(400, 200);

        shell.open();
        while (!shell.isDisposed())
        {
            if (!display.readAndDispatch())
                display.sleep();
        }
    }
}
```



```
        display.dispose();  
    }  
}
```

从图 10-1 中可以看出, 这个应用程序是可以运行的。用户可以选择标签, 以显示各种文本小部件。如果屏幕空间紧张, 并且你想以一种容易的方式堆叠窗口小部件的话, 标签文件夹是一种十分有用的控件。

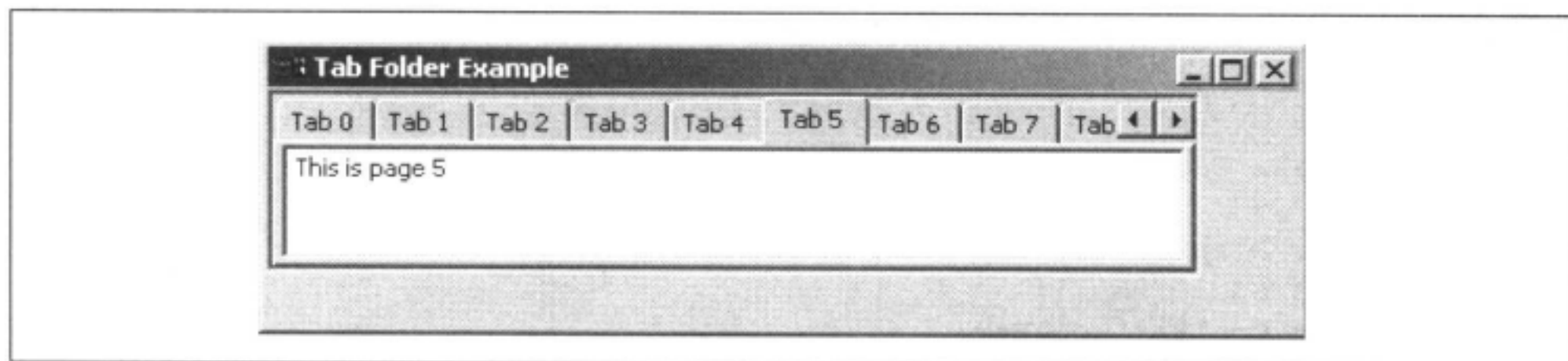


图 10-1: 标签文件夹的作用

10.2 创建 SWT 控件工具栏

问题

你想使用控件工具栏令用户拍案叫绝, 控件工具栏是能够滑动、大小可调整的工具栏。

解决方案

创建一个控件工具栏, 并添加允许用户改变位置的工具栏, 作为控件工具栏上的控件项 (cool item)。使用控件项的 `setControl` 方法将工具栏添加到每一个控件项中。

讨论

在下面这个例子 (本书代码中的 *CoolBarApp* 项目) 中, 我们将把两个活动工具栏组装到一个控件工具栏上。下面是节选的一些最有用的 *CoolBar* 方法:

```
Point computeSize(int wHint, int hHint, boolean changed)
```

返回控件工具栏的首选尺寸。

```
CoolItem getItem(int index)
```

返回当前在给定零相对索引处显示的控件项。

```
int getItemCount()
```

返回控件工具栏上包含的控件项的数量。

`CoolItem[] getItems()`

返回一个由按当前显示顺序排列的 `CoolItem` 对象组成的数组。

`Point[] getItemSizes()`

返回一个由屏幕上的点组成的数组，这些点的 `x` 和 `y` 坐标描述了控件工具栏上的控件项的宽度和高度。

创建本例中的控件工具栏是相当简单的；只需使用 `CoolBar` 构造函数并设置要用的布局即可：

```
public class CoolBarClass
{
    static Display display;
    static Shell shell;
    static CoolBar coolBar;

    public static void main(String[] args)
    {
        display = new Display();
        shell = new Shell(display);
        shell.setLayout(new GridLayout());
        shell.setText("CoolBar Example");
        shell.setSize(600, 200);

        coolBar = new CoolBar(shell, SWT.BORDER | SWT.FLAT);
        coolBar.setLayoutData(new GridData(GridData.FILL_BOTH));
        .
        .
        .
    }
}
```

下一步任务是添加包含工具栏的控件项，这将在下一节中介绍。

参考

10.3 节，在控件工具栏上添加控件项；10.4 节，在控件工具栏上添加下拉菜单。

10.3 在控件工具栏上添加控件项

问题

你已经创建了一个控件工具栏，并想在其中添加控件项。

解决方案

创建几个 `CoolItem` 对象，并把一个 `CoolBar` 对象传递给这些 `CoolItem` 对象的构造函数

数。使用CoolItem对象的setControl方法, 将ToolBar对象添加到这些CoolItem对象中。

讨论

要在控件工具栏上添加工具栏等控件项, 需要使用控件项。下面是节选的一些最有用的CoolItem方法:

```
void addSelectionListener(SelectionListener listener)
```

将该监听程序添加到监听程序集合中, 当控件项被选中时将通知该监听程序。

```
Point computeSize(int wHint, int hHint)
```

返回控件项的首选尺寸。

```
Rectangle getBounds()
```

返回一个给出控件项的大小和位置的矩形。

```
Control getControl()
```

返回控件项中包含的控件。

```
Point getMinimumSize()
```

返回控件项可以调整到的最小尺寸。

```
CoolBar getParent()
```

返回控件项的父对象, 它必须是一个CoolBar对象。

```
void setControl(Control control)
```

设置控件项中包含的控件。

```
void setSize(int width, int height)
```

设置控件项的尺寸。

继续上一节开始的示例(本书站点的CoolBarApp项目), 我们将在控件工具栏上添加两个工具栏。首先, 创建工具栏, 同时把CoolBar对象传递给工具栏的构造函数, 并创建工具栏按钮:

```
public static void main(String[] args)
{
    display = new Display();
    shell = new Shell(display);
    shell.setLayout(new GridLayout());
    shell.setText("CoolBar Example");
    shell.setSize(600, 200);

    coolBar = new CoolBar(shell, SWT.BORDER | SWT.FLAT);
    coolBar.setLayoutData(new GridData(GridData.FILL_BOTH));
}
```

```

ToolBar toolBar1 = new ToolBar(coolBar, SWT.FLAT);
for (int loopIndex = 0; loopIndex < 5; loopIndex++)
{
    ToolItem toolItem = new ToolItem(toolBar1, SWT.PUSH);
    toolItem.setText("Button " + loopIndex);
}

ToolBar toolBar2 = new ToolBar(coolBar, SWT.FLAT | SWT.WRAP);
for (int loopIndex = 5; loopIndex < 10; loopIndex++)
{
    ToolItem toolItem = new ToolItem(toolBar2, SWT.PUSH);
    toolItem.setText("Button " + loopIndex);
}

.
.
.

```

为了将这两个工具栏插入控件工具栏，在控件工具栏中新建两个 CoolItem 对象，并使用控件项的 setControl 方法将工具栏添加到新建的两个对象中：

```

public static void main(String[] args)
{
    .
    .
    .
    ToolBar toolBar2 = new ToolBar(coolBar, SWT.FLAT | SWT.WRAP);
    for (int loopIndex = 5; loopIndex < 10; loopIndex++)
    {
        ToolItem toolItem = new ToolItem(toolBar2, SWT.PUSH);
        toolItem.setText("Button " + loopIndex);
    }

    CoolItem coolItem1 = new CoolItem(coolBar, SWT.DROP_DOWN);
    coolItem1.setControl(toolBar1);

    CoolItem coolItem2 = new CoolItem(coolBar, SWT.DROP_DOWN);
    coolItem2.setControl(toolBar2);

    .
    .
    .
}

```

最后，调整工具栏的大小，以适合容纳工具栏的控件项的尺寸：

```

public static void main(String[] args)
{
    .
    .
    .
    CoolItem coolItem1 = new CoolItem(coolBar, SWT.DROP_DOWN);
    coolItem1.setControl(toolBar1);

    CoolItem coolItem2 = new CoolItem(coolBar, SWT.DROP_DOWN);
    coolItem2.setControl(toolBar2);
}

```

```
Point toolBar1Size = toolBar1.computeSize(SWT.DEFAULT, SWT.DEFAULT);
Point coolBar1Size = coolItem1.computeSize(toolBar1Size.x,
    toolBar1Size.y);
coolItem1.setSize(coolBar1Size);

Point toolBar2Size = toolBar2.computeSize(SWT.DEFAULT, SWT.DEFAULT);
Point coolBar2Size = coolItem1.computeSize(toolBar2Size.x,
    toolBar2Size.y);
coolItem2.setSize(coolBar2Size);
```

这就是所需要的全部代码，其运行结果如图 10-2 所示。每个控件项的左侧都带有一个手柄；使用鼠标拖动手柄，可以调整每个工具栏的大小，如图 10-2 所示。当然，你不必止步于简单的按钮；还可以使用能够在工具栏上显示的所有窗口小部件——详细内容请参阅第 9 章。

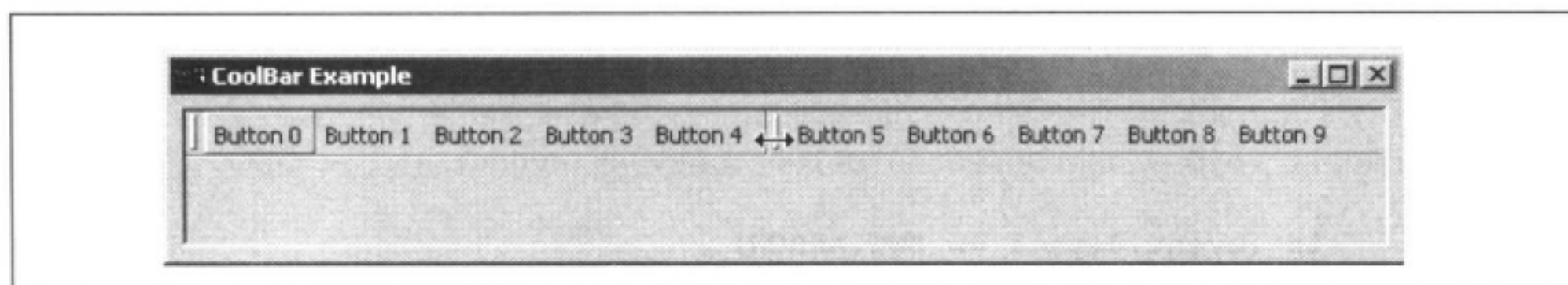


图 10-2: 一个控件工具栏中的两个工具栏

注意：这个例子的完整的代码，见 10.4 节。

参考

9.3 节，创建工具栏；9.5 节，处理工具栏事件；9.6 节，在工具栏上嵌入组合框、文本小部件和菜单；10.2 节，创建 SWT 控件工具栏；10.4 节，在控件工具栏上添加下拉菜单。

10.4 在控件工具栏上添加下拉菜单

问题

你想在控件工具栏上添加下拉菜单。

解决方案

创建自定义菜单，捕获需要处理的控件工具栏事件，并使用 Menu 类的 setLocation 和 setVisible 方法显示菜单。

注意：与在 SWT 中创建快捷菜单的方式相同。

讨论

如果将控件项的样式设置为 `SWT.DROP_DOWN`，当无法显示完整的控件项时，它将显示一个人字纹按钮，也叫做箭头按钮。单击该按钮时，可以显示一个下拉菜单，其中包含控件项中的所有按钮。

作为一个例子，我们将在上一节开发的控件工具栏上添加一个下拉菜单。为了激活菜单，应该用 `CoolBarListener` 类中的代码，捕获在控件工具栏中箭头按钮的单击事件 (`event.detail == SWT.ARROW`)：

```
class CoolBarListener extends SelectionAdapter
{
    public void widgetSelected(SelectionEvent event)
    {
        if (event.detail == SWT.ARROW)
        {
            .
            .
            .
        }
    }
}
```

在此，我们的目标是像显示菜单项一样显示按钮的标题，所以，首先应获得所单击工具栏的按钮列表：

```
class CoolBarListener extends SelectionAdapter
{
    public void widgetSelected(SelectionEvent event)
    {
        if (event.detail == SWT.ARROW)
        {
            ToolBar toolBar = (ToolBar) ((CoolItem)
                event.widget).getControl();
            ToolItem[] buttons = toolBar.getItems();
            .
            .
            .
        }
    }
}
```

接下来，创建一个菜单，其菜单项对应于按钮的标题：

```

class CoolBarListener extends SelectionAdapter
{
    public void widgetSelected(SelectionEvent event)
    {
        if (event.detail == SWT.ARROW)
        {
            ToolBar toolBar = (ToolBar) ((CoolItem)
                event.widget).getControl();
            ToolItem[] buttons = toolBar.getItems();

            if (menu != null)
            {
                menu.dispose();
            }
            menu = new Menu(coolBar);
            for (int loopIndex = 0; loopIndex < buttons.length;
                loopIndex++)
            {
                MenuItem menuItem = new MenuItem(menu, SWT.PUSH);
                menuItem.setText(buttons[loopIndex].getText());
            }
        }
    }
}

```

最后，使用 SelectionEvent 类的 x 和 y 成员确定控件工具栏的什么位置被单击了，并使用 Menu 类的 setLocation 和 setVisible 方法在单击位置显示菜单：

```

class CoolBarListener extends SelectionAdapter
{
    public void widgetSelected(SelectionEvent event)
    {
        if (event.detail == SWT.ARROW)
        {
            ToolBar toolBar = (ToolBar) ((CoolItem)
                event.widget).getControl();
            ToolItem[] buttons = toolBar.getItems();

            if (menu != null)
            {
                menu.dispose();
            }
            menu = new Menu(coolBar);
            for (int loopIndex = 0; loopIndex < buttons.length;
                loopIndex++)
            {
                MenuItem menuItem = new MenuItem(menu, SWT.PUSH);
                menuItem.setText(buttons[loopIndex].getText());
            }
        }
    }
}

```

```

        Point menuPoint = coolBar.toDisplay(new Point(event.x,
            event.y));
        menu.setLocation(menuPoint.x, menuPoint.y);
        menu.setVisible(true);
    }
}

```

在例 10-2 所示的 *CoolBarApp* 项目的最终代码中，可以看到这个新的监听程序类，*CoolBarListener*。还应注意，在代码中，我们将这个类的对象作为监听程序添加到了两个控件项中。

例 10-2: 使用 SWT 控件工具栏

```

package org.cookbook.ch10;

import org.eclipse.swt.*;
import org.eclipse.swt.events.*;
import org.eclipse.swt.layout.*;
import org.eclipse.swt.widgets.*;
import org.eclipse.swt.graphics.*;

public class CoolBarClass
{
    static Display display;
    static Shell shell;
    static CoolBar coolBar;
    static Menu menu = null;

    public static void main(String[] args)
    {
        display = new Display();
        shell = new Shell(display);
        shell.setLayout(new GridLayout());
        shell.setText("CoolBar Example");
        shell.setSize(600, 200);

        coolBar = new CoolBar(shell, SWT.BORDER | SWT.FLAT);
        coolBar.setLayoutData(new GridData(GridData.FILL_BOTH));

        ToolBar toolBar1 = new ToolBar(coolBar, SWT.FLAT);
        for (int loopIndex = 0; loopIndex < 5; loopIndex++)
        {
            ToolItem toolItem = new ToolItem(toolBar1, SWT.PUSH);
            toolItem.setText("Button " + loopIndex);
        }

        ToolBar toolBar2 = new ToolBar(coolBar, SWT.FLAT | SWT.WRAP);
        for (int loopIndex = 5; loopIndex < 10; loopIndex++)
        {
            ToolItem toolItem = new ToolItem(toolBar2, SWT.PUSH);
            toolItem.setText("Button " + loopIndex);
        }
    }
}

```

```

CoolItem coolItem1 = new CoolItem(coolBar, SWT.DROP_DOWN);
coolItem1.setControl(toolBar1);

CoolItem coolItem2 = new CoolItem(coolBar, SWT.DROP_DOWN);
coolItem2.setControl(toolBar2);

Point toolBar1Size = toolBar1.computeSize(SWT.DEFAULT, SWT.DEFAULT);
Point coolBar1Size = coolItem1.computeSize(toolBar1Size.x,
    toolBar1Size.y);
coolItem1.setSize(coolBar1Size);

Point toolBar2Size = toolBar2.computeSize(SWT.DEFAULT, SWT.DEFAULT);
Point coolBar2Size = coolItem1.computeSize(toolBar2Size.x,
    toolBar2Size.y);
coolItem2.setSize(coolBar2Size);

class CoolBarListener extends SelectionAdapter
{
    public void widgetSelected(SelectionEvent event)
    {
        if (event.detail == SWT.ARROW)
        {
            ToolBar toolBar = (ToolBar) ((CoolItem)
                event.widget).getControl();
            ToolItem[] buttons = toolBar.getItems();

            if (menu != null)
            {
                menu.dispose();
            }
            menu = new Menu(coolBar);
            for (int loopIndex = 0; loopIndex < buttons.length;
                loopIndex++)
            {
                MenuItem menuItem = new MenuItem(menu, SWT.PUSH);
                menuItem.setText(buttons[loopIndex].getText());
            }
            Point menuPoint = coolBar.toDisplay(new Point(event.x,
                event.y));
            menu.setLocation(menuPoint.x, menuPoint.y);
            menu.setVisible(true);
        }
    }
}

coolItem1.addSelectionListener(new CoolBarListener());
coolItem2.addSelectionListener(new CoolBarListener());

shell.open();
while (!shell.isDisposed())
{
    if (!display.readAndDispatch())
        display.sleep();
}

```

```
        display.dispose();  
    }  
}
```

运行结果如图 10-3 所示；当调整控件项的大小使第一个按钮的部分被遮挡住时，单击出现的箭头按钮，可以显示图 10-3 所示的下拉菜单。相当不错！

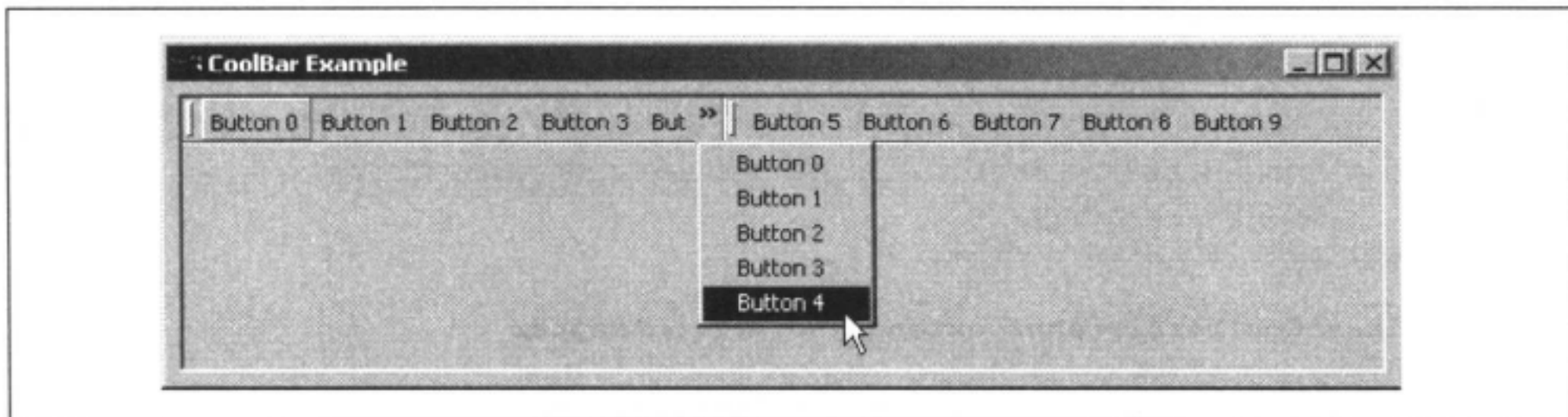


图 10-3：在控件工具栏上添加一个下拉菜单

参考

10.2 节，创建 SWT 控件工具栏；10.3 节，在控件工具栏上添加控件项。

10.5 创建 SWT 树

问题

你需要以分层次的、可折叠、可展开的形式显示数据项。

解决方案

使用基于 `Tree` 和 `TreeItem` 类的 SWT 树小部件。

讨论

作为一个例子，我们将创建一棵包含多层树项的树（本书站点的 *TreeApp* 项目）。下面是节选的一些有用的 `Tree` 类的方法：

```
void addSelectionListener(SelectionListener listener)
```

将该监听程序添加到监听程序集合中，当树中的选择改变时将通知该监听程序。

```
void deselectAll()
```

取消对树中所有选定项的选择。


```
TreeItem[] getItems()
```

返回一个由树中包含的所有树项组成的数组。

```
TreeItem[] getSelection()
```

返回一个由树中选定的 `TreeItem` 对象组成的数组。

```
int getSelectionCount()
```

返回树中选定项的数量。

```
void selectAll()
```

选择树中的所有项。

```
void setSelection(TreeItem[] items)
```

将树中选定的项设置成为给定数组的元素。

TreeApp 示例的代码以下面这种方式创建一棵树：

```
final Tree tree = new Tree(shell, SWT.BORDER | SWT.V_SCROLL |  
    SWT.H_SCROLL);  
tree.setSize(290, 260);
```

以这种方式添加到树中的项目是 `TreeItem` 类的对象。下面是节选的一些 `TreeItem` 类的方法：

```
boolean getChecked()
```

如果这个树项被选中，返回 `true`，否则返回 `false`。

```
boolean getGrayed()
```

如果这个树项变为不可用，返回 `true`，否则返回 `false`。

```
int getItemCount()
```

返回这个树项中包含的子项的数量。

```
TreeItem[] getItems()
```

返回一个由 `TreeItem` 对象组成的数组，这些对象是该树项的子项。

```
void setChecked(boolean checked)
```

设置该树项的复选状态。

```
void setExpanded(boolean expanded)
```

设置该树项的展开状态。

```
void setGrayed(boolean grayed)
```

设置该树项的变灰（不可用）状态。

```
void setImage(Image image)
```

设置该树项的图像。

```
void setText(String string)
```

设置该树项的文本。

在树中添加项时，需要把要添加项的父项传递给新加项的构造函数。为了在树小部件中添加 10 个树项，可以使用如下的代码：

```
final Tree tree = new Tree(shell, SWT.BORDER | SWT.V_SCROLL |
    SWT.H_SCROLL);
tree.setSize(290, 260);

for (int loopIndex0 = 0; loopIndex0 < 10; loopIndex0++)
{
    TreeItem treeItem0 = new TreeItem(tree, 0);
    treeItem0.setText("Level 0 Item " + loopIndex0);
    .
    .
    .
}
```

要在已有的树项中添加子项，可把已有树项传递给子项的构造函数，向下增加任意多层。在本例中，我们将给每一个顶层树项添加 10 个子项，再给每个子项添加十多个子项：

```
final Tree tree = new Tree(shell, SWT.BORDER | SWT.V_SCROLL |
    SWT.H_SCROLL);
tree.setSize(290, 260);

for (int loopIndex0 = 0; loopIndex0 < 10; loopIndex0++)
{
    TreeItem treeItem0 = new TreeItem(tree, 0);
    treeItem0.setText("Level 0 Item " + loopIndex0);
    for (int loopIndex1 = 0; loopIndex1 < 10; loopIndex1++)
    {
        TreeItem treeItem1 = new TreeItem(treeItem0, 0);
        treeItem1.setText("Level 1 Item " + loopIndex1);
        for (int loopIndex2 = 0; loopIndex2 < 10; loopIndex2++)
        {
            TreeItem treeItem2 = new TreeItem(treeItem1, 0);
            treeItem2.setText("Level 2 Item " + loopIndex2);
        }
    }
}
```

这将创建一棵如图 10-4 所示的树。

关于树的进一步介绍，如处理选择事件，请参阅下一节。

注意：关于这个示例的完整代码，参见 10.6 节。

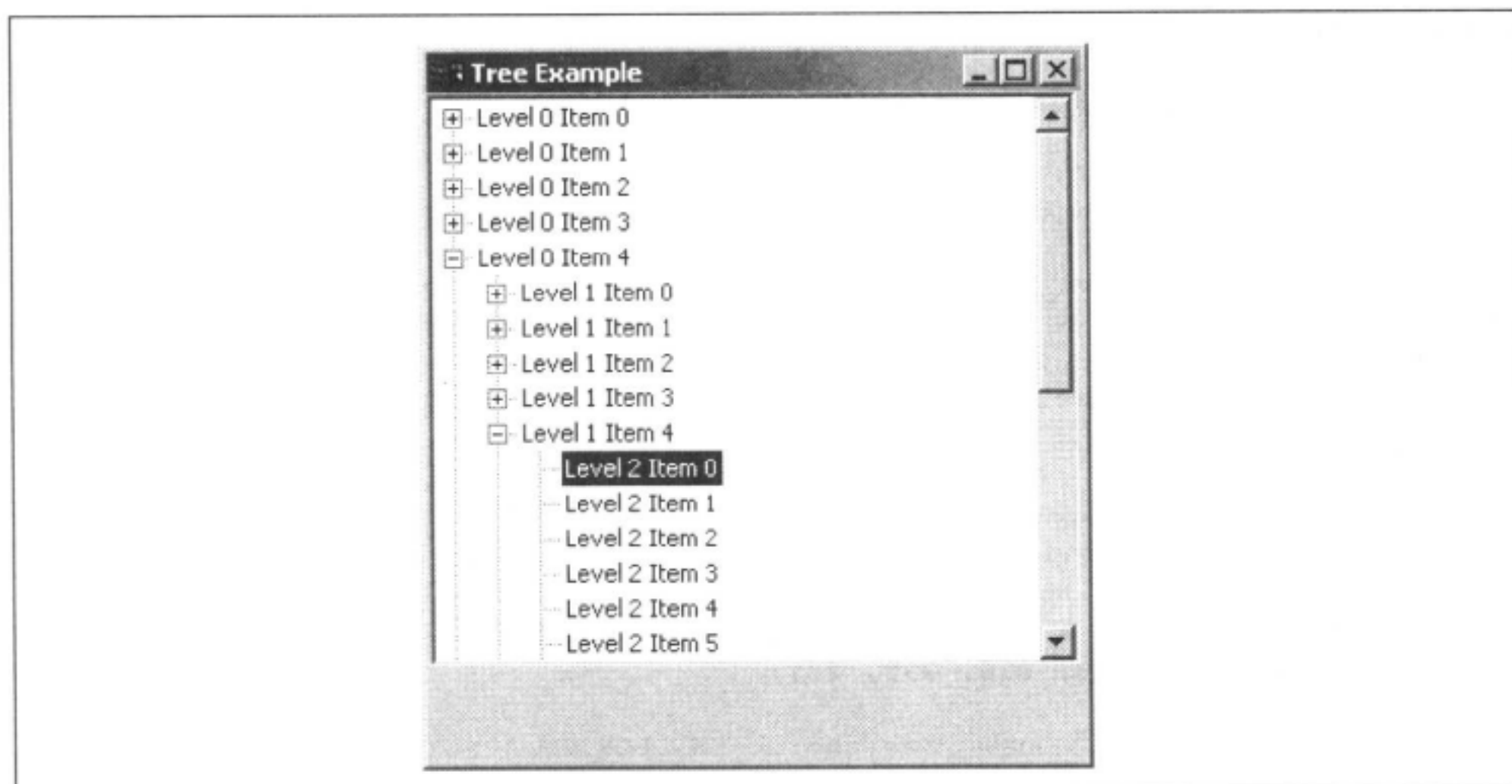


图 10-4: SWT 树

参考

10.6 节, 处理树事件; 10.7 节, 在树项中添加复选框; 10.8 节, 在树项中添加图像; *Eclipse* (O'Reilly) 一书的第 8 章。

10.6 处理树事件

问题

你需要知道用户何时在树小部件中选择一个树项。

解决方案

使用 `addListener` 或 `addSelectionListener` 方法, 在 `Tree` 小部件中添加一个新的监听程序。

讨论

作为一个例子, 我们将在上一节开发的树中添加一个监听程序。另外, 还将添加一个文本小部件, 以显示用户的选择, 如例 10-3。

例 10-3: SWT 树

```
package org.cookbook.ch10;

import org.eclipse.swt.*;
import org.eclipse.swt.widgets.*;

public class TreeClass
{
    public static void main(String[] args)
    {
        Display display = new Display();
        Shell shell = new Shell(display);
        shell.setText("Tree Example");

        final Text text = new Text(shell, SWT.BORDER);
        text.setBounds(0, 270, 290, 25);

        final Tree tree = new Tree(shell, SWT.BORDER | SWT.V_SCROLL |
            SWT.H_SCROLL);
        tree.setSize(290, 260);
        shell.setSize(300, 330);

        for (int loopIndex0 = 0; loopIndex0 < 10; loopIndex0++)
        {
            TreeItem treeItem0 = new TreeItem(tree, 0);
            treeItem0.setText("Level 0 Item " + loopIndex0);
            for (int loopIndex1 = 0; loopIndex1 < 10; loopIndex1++)
            {
                TreeItem treeItem1 = new TreeItem(treeItem0, 0);
                treeItem1.setText("Level 1 Item " + loopIndex1);
                for (int loopIndex2 = 0; loopIndex2 < 10; loopIndex2++)
                {
                    TreeItem treeItem2 = new TreeItem(treeItem1, 0);
                    treeItem2.setText("Level 2 Item " + loopIndex2);
                }
            }
        }

        tree.addListener(SWT.Selection, new Listener()
        {
            public void handleEvent(Event event)
            {
                text.setText(event.item + " was selected");
            }
        });

        shell.open();
        while (!shell.isDisposed())
        {
            if (!display.readAndDispatch())
                display.sleep();
        }
        display.dispose();
    }
}
```

```
}  
}
```

使用显示选择事件的新加的监听程序和文本小部件, *TreeApp* 示例可以显示用户做出的选择, 如图 10-5。

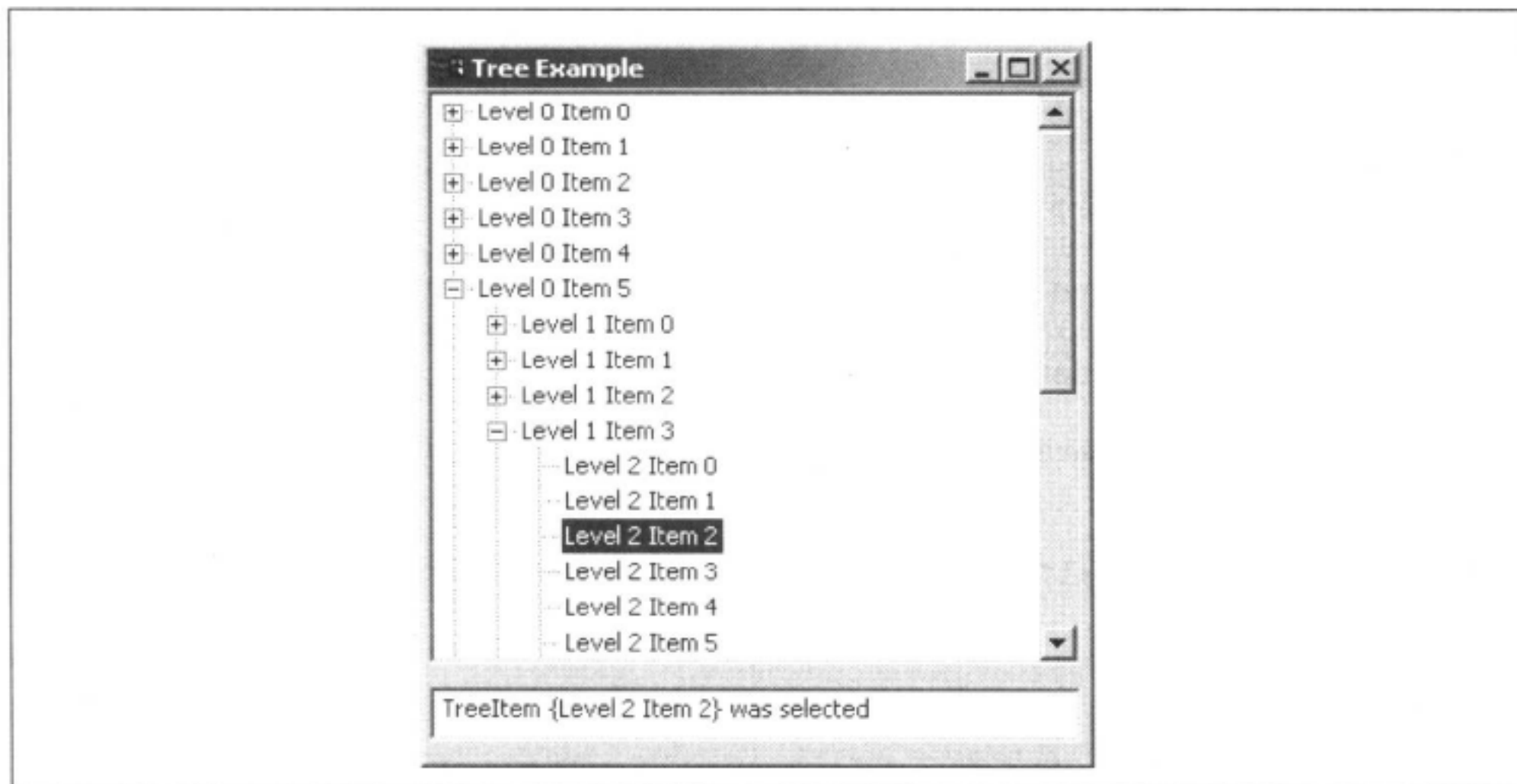


图 10-5: 捕获树选择事件

参考

10.5 节, 创建 SWT 树; 10.7 节, 在树项中添加复选框; 10.8 节, 在树项中添加图像。

10.7 在树项中添加复选框

问题

你需要使用户能够选择树中的项, 而且当一些项被选中时, 使这些项保持选中状态, 直至明确地取消选择为止。

解决方案

通过添加 `SWT.CHECK` 样式确保 SWT 树支持复选框, 并使监听程序监听 `SWT.CHECK` 事件。

讨论

作为一个例子，可在前两节开发的树中添加复选框（本书站点的 *TreeApp* 项目）。要添加复选框，应在创建树小部件时将 `SWT.CHECK` 样式添加到其中：

```
final Tree tree = new Tree(shell, SWT.CHECK | SWT.BORDER | SWT.V_SCROLL |  
    SWT.H_SCROLL);
```

然后通过监听程序监听 `SWT.CHECK` 事件。其代码如下所示：

```
tree.addListener(SWT.Selection, new Listener()  
{  
    public void handleEvent(Event event)  
    {  
        if (event.detail == SWT.CHECK)  
        {  
            text.setText(event.item + " was checked.");  
        } else  
        {  
            text.setText(event.item + " was selected");  
        }  
    }  
});
```

运行结果如图 10-6 所示，其中每一个树项都有一个复选框，并监测到了复选框事件。

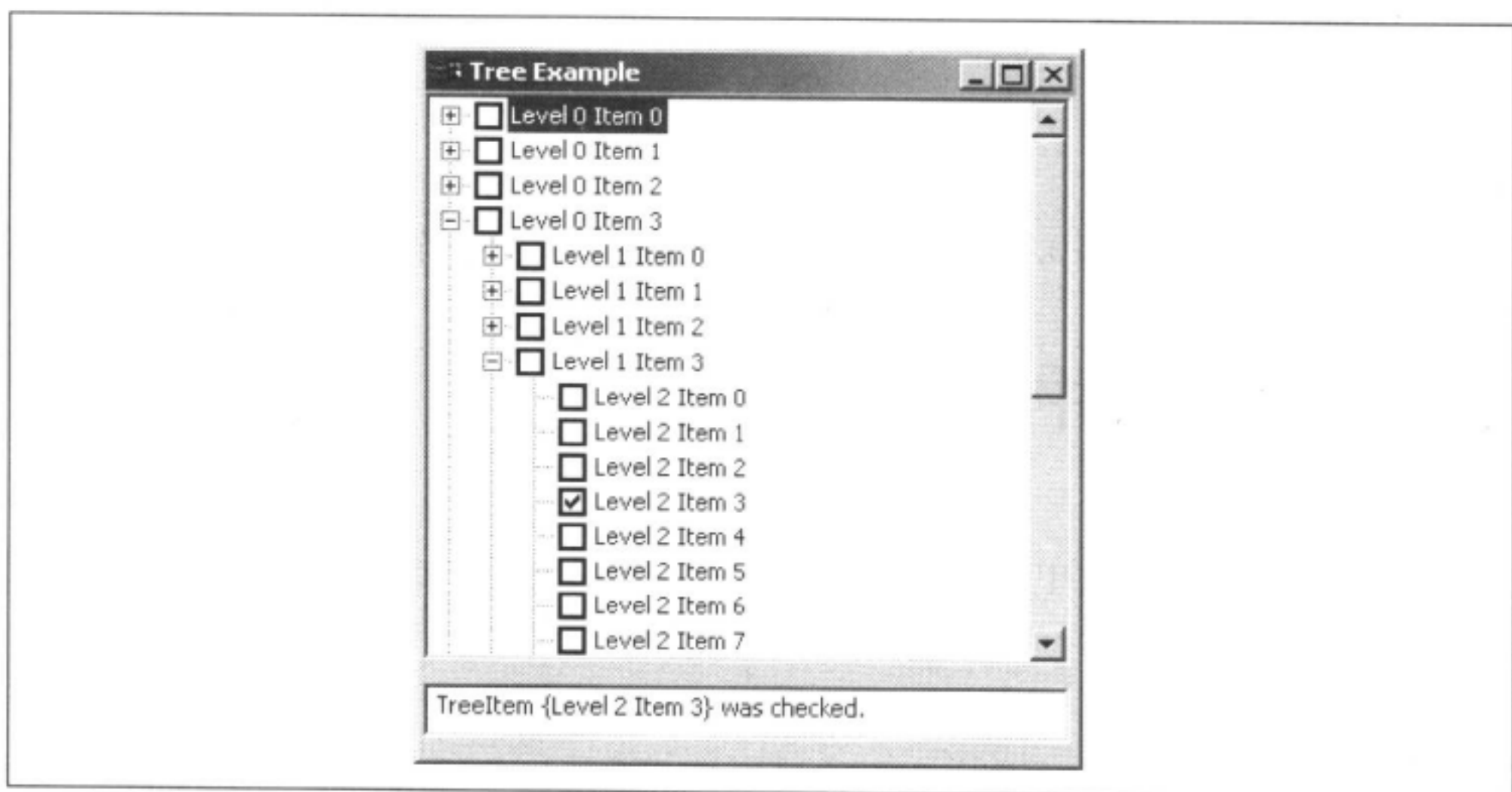


图 10-6：在 SWT 树中支持复选标记

注意：也可以使用 `setChecked` 和 `getChecked` 方法处理复选标记。

参考

10.5 节, 创建 SWT 树; 10.6 节, 处理树事件; 10.8 节, 在树项中添加图像。

10.8 在树项中添加图像

问题

你想在树项中添加图像。

解决方案

使用 `TreeItem` 类的 `setImage` 方法和 `getImage` 方法。

讨论

要在树项中添加图像, 可向 `setImage` 方法传递一个新的 `Image` 对象。要了解如何创建 `Image` 对象, 可参阅 9.9 节。使用 `getImage` 方法, 可以获得树项的图像。

参考

9.9 节, 创建图像菜单项; 10.5 节, 创建 SWT 树; 10.6 节, 处理树事件; 10.7 节, 在树项中添加复选框。

10.9 创建 SWT 浏览器小部件

问题

你想为用户提供某种联机访问, 例如, 访问联机帮助。

解决方案

使用 SWT 浏览器小部件。从 Eclipse 3.0 开始, SWT 包含一个内置的浏览器小部件。

讨论

在 Eclipse 2.x 中, 在 SWT 窗口中显示浏览器方面, 并没有太多的选择。事实上, 只能在 Windows 中使用 OLE 才能显示浏览器。下面这个例子说明了如何在 SWT 窗口中显示浏览器。创建一个 `OleFrame` 对象, 并将系统的浏览器连接至这个对象:

```

import org.eclipse.swt.ole.win32.*;
.
.
.
OleControlSite oleControlSite;

OleFrame oleFrame = new OleFrame(shell, SWT.NONE);
oleControlSite = new OleControlSite(oleFrame, SWT.NONE, "Shell.Explorer");
oleControlSite.doVerb(OLE.OLEIVERB_INPLACEACTIVATE);
shell.open();
.
.
.

```

然后创建一个 OleAutomation 对象，并浏览到所需的 URL，如下所示：

```

import org.eclipse.swt.ole.win32.*;
.
.
.
OleControlSite oleControlSite;

OleFrame oleFrame = new OleFrame(shell, SWT.NONE);
oleControlSite = new OleControlSite(oleFrame, SWT.NONE, "Shell.Explorer");
oleControlSite.doVerb(OLE.OLEIVERB_INPLACEACTIVATE);
shell.open();

final OleAutomation browser = new OleAutomation(oleControlSite);

int[] browserIDs = browser.getIDsofNames(new String[]{"Navigate", "URL"});
Variant[] address = new Variant[] {new Variant("http://www.oreilly.com")};
browser.invoke(browserIDs[0], address, new int[]{browserIDs[1]});

```

Eclipse 3.0

在 Eclipse 3.x 中，事情变得更容易了，这是因为 SWT 自带了一个内置的浏览器小部件。这个小部件目前得到了 Windows（使用 Internet Explorer 5 以上版本）和 Linux GTK（使用 Mozilla 1.4 GTK2）的支持。最终会有更多的操作系统支持浏览器小部件。下面是这个小部件的一些有用的导航方法：

```
boolean back()
```

后退一页。

```
boolean forward()
```

前进一页。

```
boolean setUrl(String string)
```

导航到一个 URL。

```
void stop()
```

停止当前操作。

我们将创建一个示例（本书站点的 *BrowserApp* 项目），以说明如何使用这个小部件。这个示例包含一个工具栏，其中有 Go、Back 和 Stop 三个按钮；示例中还包含一个文本小部件，可以在其中输入 URL：

```
public class BrowserClass {
    public static void main(String[] args) {
        Display display = new Display();
        final Shell shell = new Shell(display);
        shell.setText("Browser Example");
        shell.setSize(620, 500);

        ToolBar toolbar = new ToolBar(shell, SWT.NONE);
        toolbar.setBounds(5, 5, 200, 30);

        ToolItem goButton = new ToolItem(toolbar, SWT.PUSH);
        goButton.setText("Go");

        ToolItem backButton = new ToolItem(toolbar, SWT.PUSH);
        backButton.setText("Back");

        ToolItem stopButton = new ToolItem(toolbar, SWT.PUSH);
        stopButton.setText("Stop");

        final Text text = new Text(shell, SWT.BORDER);
        text.setBounds(5, 35, 400, 25);
```

在 Eclipse 3.x 中创建这个浏览器相当简单：

```
final Browser browser = new Browser(shell, SWT.NONE);
browser.setBounds(5, 75, 600, 400);
```

下面创建的监听程序将调用相应的浏览器方法，来处理 Go、Back 和 Stop 按钮事件，如下所示：

```
Listener listener = new Listener() {
    public void handleEvent(Event event) {
        ToolItem item = (ToolItem) event.widget;
        String string = item.getText();
        if (string.equals("Back"))
            browser.back();
        else if (string.equals("Stop"))
            browser.stop();
        else if (string.equals("Go"))
            browser.setUrl(text.getText());
    }
};
```

余下的工作就是为文本小部件添加一个监听程序，在用户按下 Enter 键时，这个监听程序浏览到用户输入的 URL，并打开 shell，以显示新建的浏览器。有关代码如例 10-4 所示。

例 10-4：使用 SWT 浏览器小部件

```
package org.cookbook.ch10;

import org.eclipse.swt.*;
import org.eclipse.swt.widgets.*;
import org.eclipse.swt.browser.*;

public class BrowserClass {
    public static void main(String[] args) {
        Display display = new Display();
        final Shell shell = new Shell(display);
        shell.setText("Browser Example");
        shell.setSize(620, 500);

        ToolBar toolbar = new ToolBar(shell, SWT.NONE);
        toolbar.setBounds(5, 5, 200, 30);

        ToolItem goButton = new ToolItem(toolbar, SWT.PUSH);
        goButton.setText("Go");

        ToolItem backButton = new ToolItem(toolbar, SWT.PUSH);
        backButton.setText("Back");

        ToolItem stopButton = new ToolItem(toolbar, SWT.PUSH);
        stopButton.setText("Stop");

        final Text text = new Text(shell, SWT.BORDER);
        text.setBounds(5, 35, 400, 25);

        final Browser browser = new Browser(shell, SWT.NONE);
        browser.setBounds(5, 75, 600, 400);

        Listener listener = new Listener() {
            public void handleEvent(Event event) {
                ToolItem item = (ToolItem) event.widget;
                String string = item.getText();
                if (string.equals("Back"))
                    browser.back();
                else if (string.equals("Forward"))
                    browser.forward();
                else if (string.equals("Stop"))
                    browser.stop();
                else if (string.equals("Go"))
                    browser.setUrl(text.getText());
            }
        };
    }
}
```



```
backButton.addListener(SWT.Selection, listener);
stopButton.addListener(SWT.Selection, listener);

text.addListener(SWT.DefaultSelection, new Listener() {
    public void handleEvent(Event e) {
        browser.setUrl(text.getText());
    }
});

shell.open();
browser.setUrl("http://oreilly.com");
while (!shell.isDisposed()) {
    if (!display.readAndDispatch())
        display.sleep();
}
display.dispose();
}
```

新建的浏览器如图 10-7 所示，我们使用这个浏览器浏览到了 O'Reilly 的 Web 站点。用户可以在地址文本小部件中输入 URL，工具栏上的按钮也是有效的。这就是 Eclipse 3.0 中的浏览器小部件。非常棒！

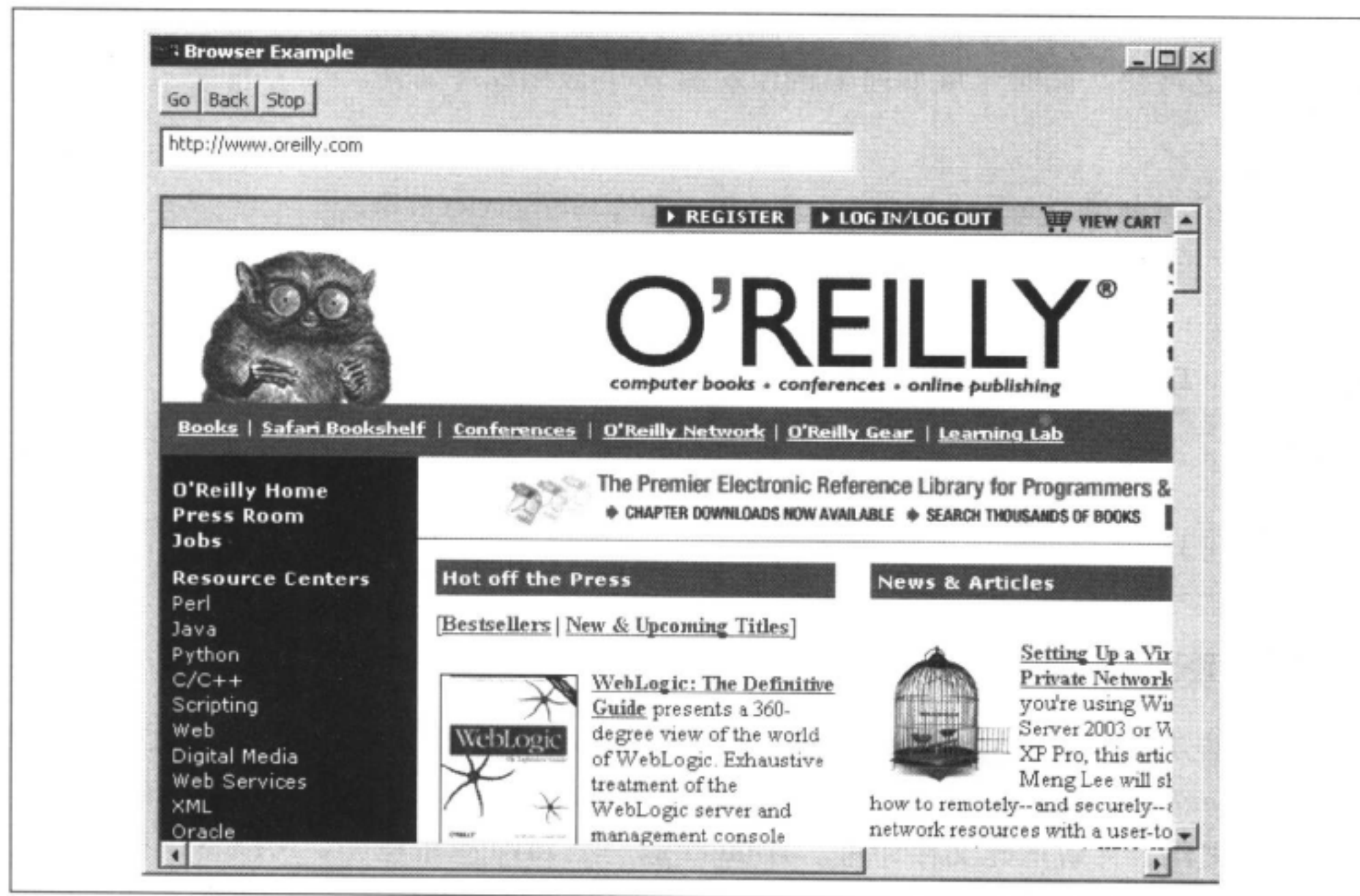


图 10-7: SWT 浏览器小部件

第 11 章

JSP、Servlet 和 Eclipse

11.0 简介

本章讨论 Eclipse 在 Web 开发中的应用，特别是使用 JavaServer Pages (JSP) 和 Java Servlet 进行开发时。本章将使用 JSP 和 Servlet 的参考 Web 服务器——Tomcat Web 服务器，该服务器可以从 Apache 的 Web 站点免费获得。在安装并启动了 Tomcat 之后，将 Eclipse 与之连接，同时了解如何编辑和安装 Web 应用程序文件。

注意： 尽管本章使用 Tomcat，但所有内容适用于任何 JSP/Servlet 容器。

11.1 安装 Tomcat

问题

你需要访问 JSP 和 Servlet Web 容器。

解决方案

下载并安装 Tomcat Web 服务器，这是 JSP 和 Servlet 的参考安装。从站点 <http://jakarta.apache.org/tomcat/> 免费获得 Tomcat。安装是简单的，只需下载适合你的操作系统的压缩文件，并解压此文件即可。Tomcat 是一个 Java 应用程序，所以不需要专门的安装程序。

讨论

如果你无法访问 JSP/Servlet Web 服务器，可以从站点 <http://jakarta.apache.org/tomcat/> 下载 Tomcat；只需下载适合你的操作系统的压缩文件即可。Tomcat 的好处之一就是，在安装它时所需要做的全部工作只是解压下载的文件。Tomcat 是一个 Java 应用程序，所以它可以在你的 Java 安装上运行（进一步的内容请参阅 11.2 节）。

我们将使用本书编写时的 Tomcat 的最新版本，即 5.0.19 版。安装非常简单，只需下载并解压它即可。下面是安装后的目录结构：

jakarta-tomcat-5.0.19	
__bin	二进制文件
__common	代码和 Web 应用程序使用的类
__classes	通用 Java 类
__endorsed	签注的 Java 类
__lib	.jar 格式的通用 Java 类
__conf	配置文件
__logs	服务器的日志文件
__server	Tomcat 内部类
__shared	共享文件
__temp	临时文件
__webapps	Web 应用程序使用的目录
__work	保存临时文件的临时目录

这里，*webapps* 目录是一个重要的目录，至少从编写 Web 应用程序的角度来讲是这样，这是 JSP 和 Servlet 的安装目录。

在本章中，我们将把示例存储在一个名为 *ch11* 的目录中，它是 Tomcat *webapps* 目录的子目录。注意，这个新的目录本身必须有一个名为 *WEB-INF* 的子目录，而后者必须有两个子目录，*classes* 和 *lib*：

webapps	
__ch11	存储第 11 章示例的文件夹
__WEB-INF	关于第 11 章的 Web 应用程序的信息
__classes	第 11 章的 Web 应用程序使用的 Java 类
__lib	第 11 章的 Web 应用程序使用的 JAR 文件

注意，尽管这里的 *WEB-INF*、*classes* 和 *lib* 目录是空的，但这些目录是必须有的，否则 Tomcat 将无法为浏览器提供文件。本章的示例将存储在 *ch11* 目录中。

参考

11.2 节，启动 Tomcat；*Tomcat: the Definitive Guide* (O'Reilly) 一书的第 1 章；*JavaServer Page* (O'Reilly) 一书的第 4 章；*Eclipse* (O'Reilly) 一书的第 9 章。

11.2 启动 Tomcat

问题

你想让 Tomcat 运行起来。

解决方案

设置 `JAVA_HOME` 和 `CATALINA_HOME` 环境变量，把目录修改为 Tomcat 的 `bin` 目录，然后在 Windows 中输入 `startup`，或者在 Unix 中运行 `startup.sh`。

讨论

在从命令行启动 Tomcat 之前，必须设置这两个环境变量：

`JAVA_HOME`

将这个环境变量设置为 Java 的安装目录，即 Java `bin` 目录的父目录；例如，在 Windows 中为 `C:\jdk1.4`，在 Unix 中为 `/usr/java`。

`CATALINA_HOME`

将这个环境变量设置为 Tomcat 的安装目录，即 Tomcat `bin` 目录的父目录；例如，在 Windows 中为 `C:\tomcat\jakarta-tomcat-5.0.19`。

可以从命令提示符（在 Windows 就是 DOS 提示符，在 Unix 中是 shell）设置这两个环境变量，例如：

```
set JAVA_HOME=C:\jdk1.4
```

具体的设置随操作系统的不同而有所变化。例如，在 Unix `tcsh` shell 中，使用 `setenv`。在设置了这两个环境变量之后，在 Windows 中通过将目录改变为 Tomcat 的 `bin` 目录并键入 `startup`，或者在 Unix 中运行 `startup.sh`，可以启动 Tomcat。注意，在 Windows 中，会出现一个新的 DOS 窗口，显示初始化消息。要关闭 Tomcat，在 Windows 中键入 `shutdown`，或者在 Unix 命令行运行 `shutdown.sh` 即可。

现在，Tomcat 已经启动，打开一个浏览器，并导航到 `http://localhost:8080`，这将打开 Tomcat 的欢迎页，如图 11-1 所示。这个 URL 中 `localhost` 部分是在你的本地机器上安装 Web 服务器的目录（对应的 IP 地址为 `127.0.0.1`），而 `8080` 是端口号（Web 服务器通常使用端口 `80`，但 Tomcat 使用 `8080`，以避免与其他服务器冲突）。

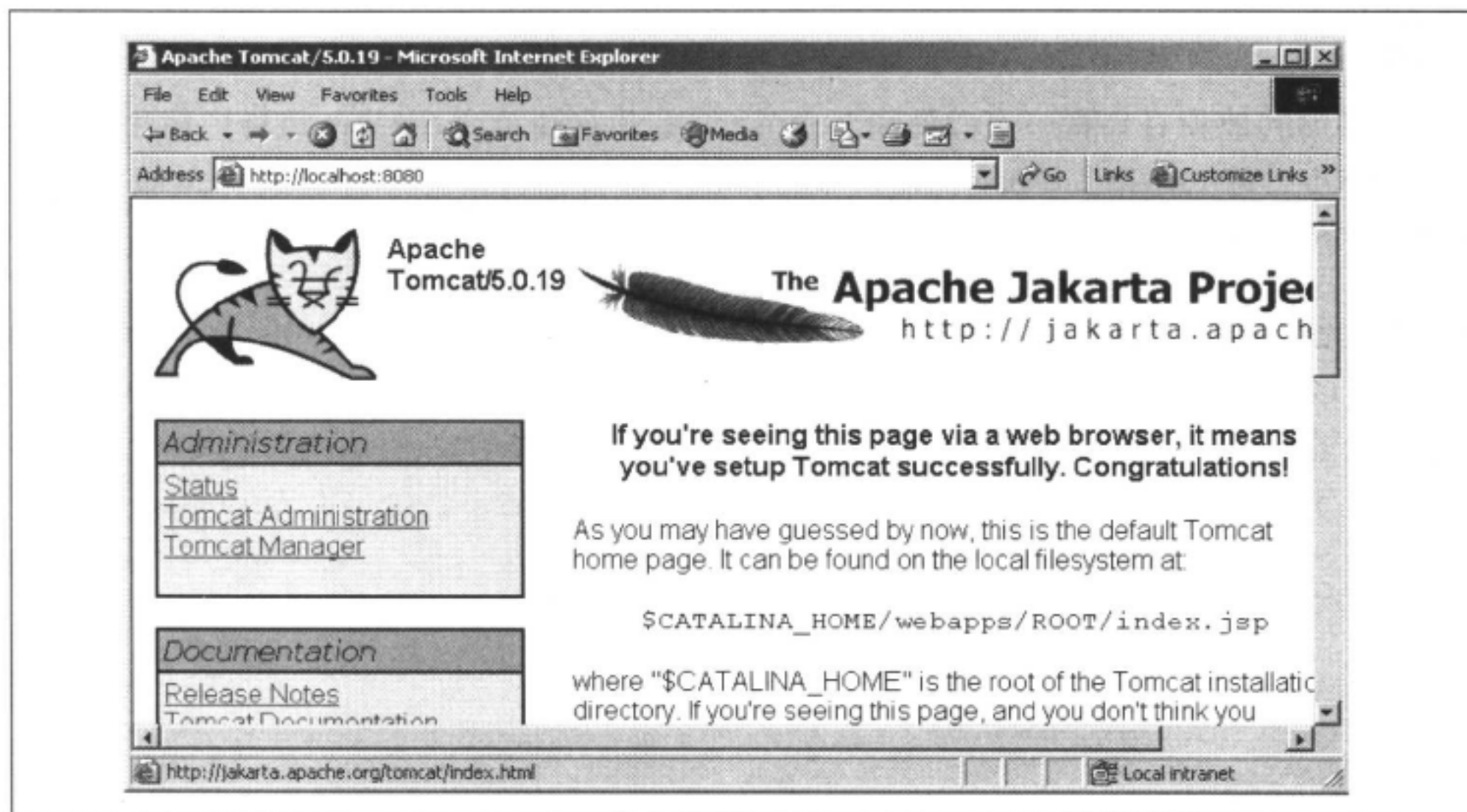


图 11-1: Tomcat 5.0.19 启动页

注意： 当在 Tomcat 的目录中存储了新的 `.class` 文件，或者编辑了随后将看到的 `web.xml` 等文件之后，应该关闭 Tomcat 并重新启动它。这样做可以确保 Tomcat 把所做的修改从 `webapps` 目录复制到 Tomcat 的工作目录，这是实际为浏览器提供文档的目录（可以通过几种方式配置 Tomcat，使它不必重新启动，但这超出了本书的范围）。

参考

11.1 节，安装 Tomcat；*Tomcat: the Definitive Guide* (O'Reilly) 一书的第 1 章；*JavaServer Page* (O'Reilly) 一书的第 4 章；*Eclipse* (O'Reilly) 一书的第 9 章。

11.3 创建 JSP 文件

问题

你想编写 JSP 文件，并在一个 Web 浏览器中查看这些文件。

解决方案

在 Eclipse 创建并编辑 JSP 文件。

讨论

与对待任何其他项目一样，可以使用 Eclipse 编写并存储可在 Servlet 容器中使用的 *.jsp* 文件。至少，可以使用 Eclipse 创建这些文件，然后再把这些文件复制到 Web 服务器安装目录下的正确目录中。

JSP 文件是能够把 Java 代码存储在 `scriptlet`、`declaration` 和 `expression` 元素中的文件。在这些元素中，`scriptlets` 元素最常用，而且它可以包含一般的 Java 代码。`scriptlet` 元素包含在标记 `<%` 和 `%>` 中，如例 11-1 所示。在这个例子中，使用内置的 `out` 对象的 `println` 方法来显示 JSP 页面在浏览器中输出的文本 “This JSP is functional.”。

例 11-1：一个示例 JSP

```
<HTML>
  <HEAD>
    <TITLE>JSP Sample</TITLE>
  </HEAD>

  <BODY>
    <H1>JSP Sample</H1>
    <% out.println("This JSP is functional."); %>
  </BODY>
</HTML>
```

创建这个 JSP 文件的一种简单的方式是将它输入 Eclipse，如图 11-2 所示，在这里，我们创建了一个新的项目 *JSP*，新建了一个文件 *greeting.jsp*，用于存储 JSP 代码。这里不能进行语法检查；Eclipse 仅仅使用了其标准的、默认的编辑器。

注意： 如果想对 JSP 文档进行语法检查，可以尝试使用一种 XML 编辑器，即 XML Buddy。该编辑器可以从站点 <http://www.xmlbuddy.com> 免费获得。

下一步是把 *greeting.jsp* 文件安装到 Tomcat 的安装目录中；在本例中，只需把这个文件复制到本章示例使用的目录中，即 *webapps\ch11* 目录中即可。

现在重新启动 Tomcat，打开一个 Web 浏览器，并导航到 <http://localhost:8080/Ch11/greeting.jsp>。应该可以看到如图 11-3 所示的结果。系统内部的真相是，Tomcat 把 JSP 文件编译成 Servlet 形式并运行它，得到如图 11-3 所示的结果。

这就是创建 JSP 文件的过程。尽管 Eclipse 在此只是起到了一个文本编辑器的作用，但它的功能远不止这些。详细内容请参阅接下来的几节。

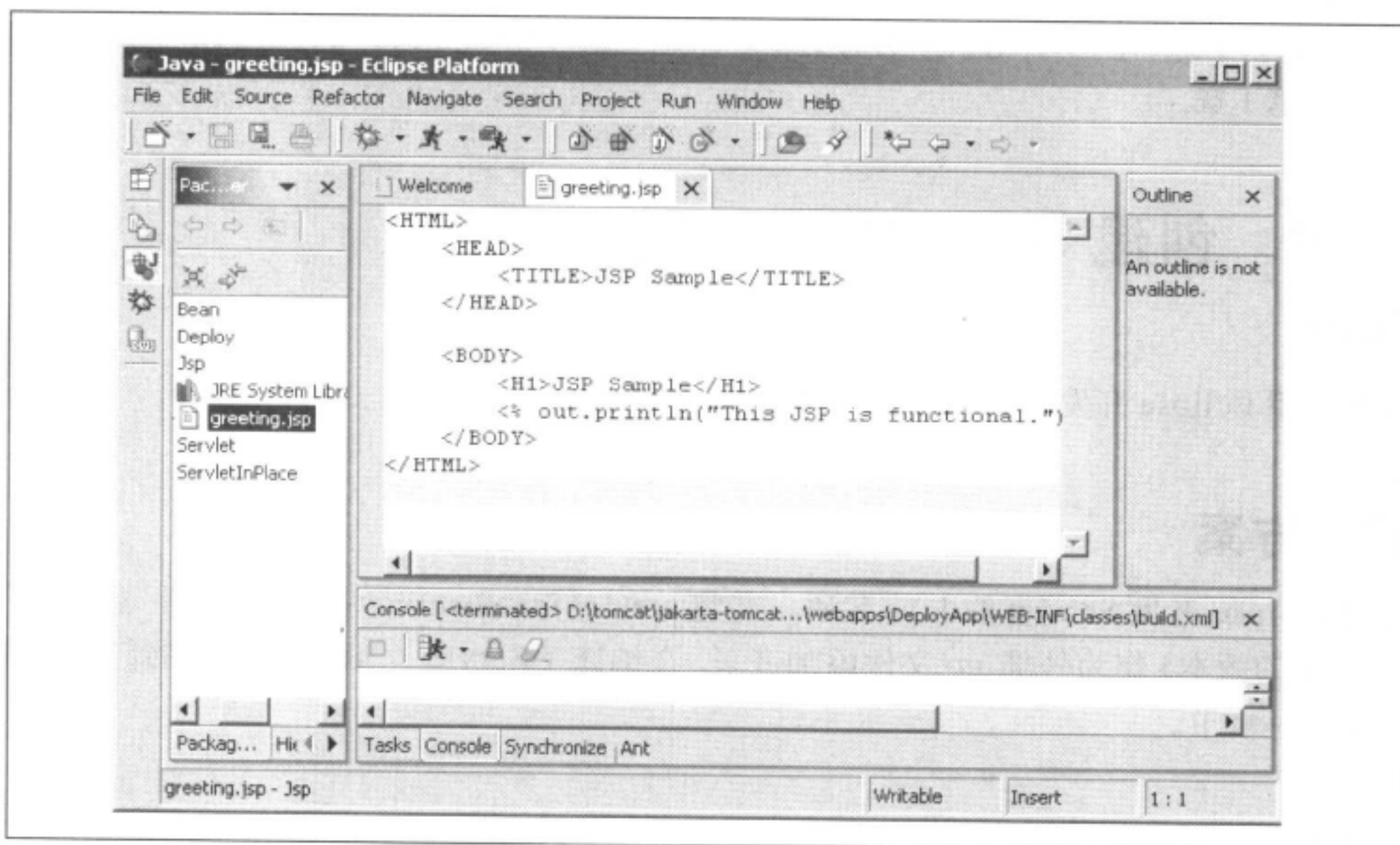


图 11-2: 编辑 JSP 代码

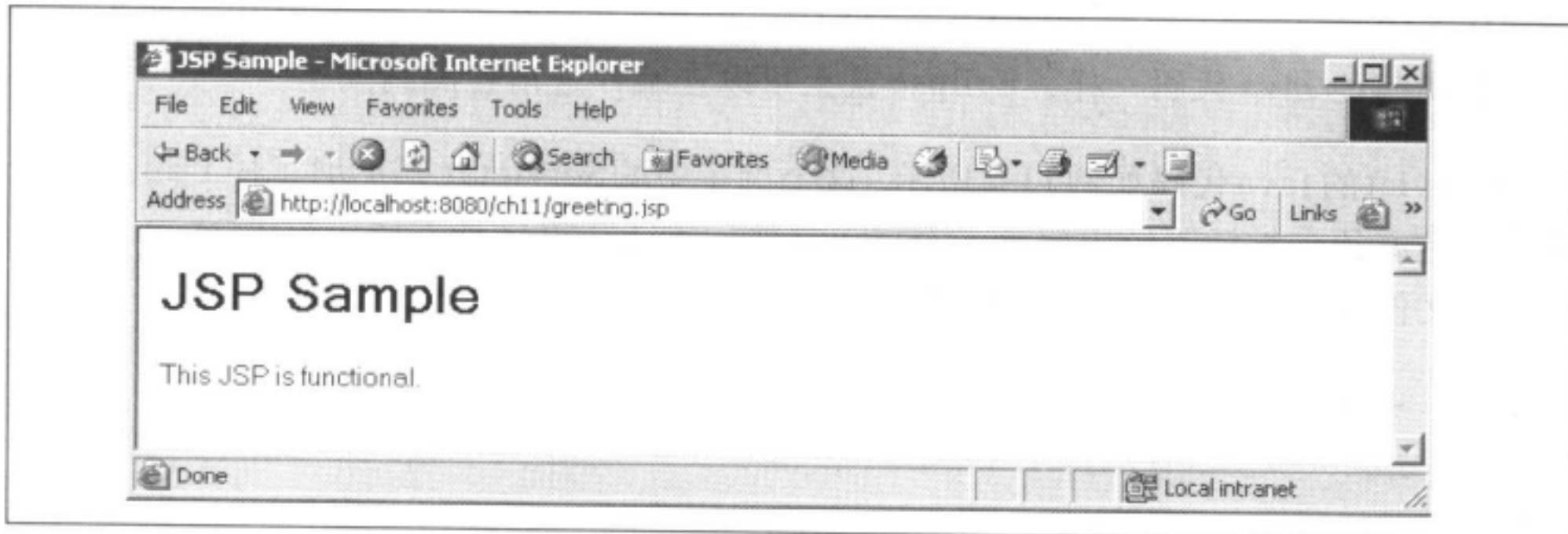


图 11-3: 一个可运行的 JSP 文件

Eclipse 3.0

Eclipse 3.0 计划针对 JSP 文件增加更多的语法检查, 但直到编写本书为止, 这种支持尚未实现。如果实现了对 JSP 代码的语法检查支持, 那就太好了。

参考

11.1 节, 安装 Tomcat; 11.7 节, 就地编辑 *web.xml* 文件; 11.9 节, 连接到 Java 组件;

JavaServer Page (O'Reilly) 一书的第 3 章, *Java Servlet and JSP Cookbook* (O'Reilly) 一书的第 1 章。

11.4 创建 Servlet

问题

你想使用 Eclipse 开发 Servlet。

解决方案

使用 Eclipse 开发 Servlet 的 Java 代码, 并将 *servlet.jar* 或 *servlet-api.jar* 文件 (取决于 Tomcat 的版本) 作为外部 *.jar* 文件添加进来。在编译了 Servlet 之后, 将其复制到 Tomcat 的安装目录中。

讨论

导入上一节开发的那种 JSP 文件, 使得联机 Java 编程变得更加容易。JSP 文件实际上被编译成 Servlet 代码, 即纯 Java 代码, 没有混入任何 HTML。在 Eclipse 中开发 Servlet 就像开发其他 Java 代码一样: Eclipse 甚至可以在编译之前检测到问题。

一个 Servlet 的 Java 代码如例 11-2 所示。这个例子中的 Servlet 代码扩展了 *HttpServlet* 类, 而且必须在编译路径中包含一个 *.jar* 文件, 才能获得这种支持。在 Tomcat 4.x 中, 该 *.jar* 文件是 *servlet.jar*; 在 Tomcat 5.x 中, 是 *servlet-api.jar*。

例 11-2: 一个简单的 Servlet

```
package org.cookbook.ch11;

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class ServletClass extends HttpServlet
{
    public void doGet(HttpServletRequest request,
        HttpServletResponse response)
        throws IOException, ServletException
    {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();

        out.println("<HTML>");
        out.println("<HEAD>");
```

```
        out.println("<TITLE>");
        out.println("Servlet Sample");
        out.println("</TITLE>");
        out.println("</HEAD>");
        out.println("<BODY>");
        out.println("<H1>");
        out.println("Servlet Sample");
        out.println("</H1>");
        out.println("This servlet is functional.");
        out.println("</BODY>");
        out.println("</HTML>");
    }
}
```

在这个例子中，代码将显示以下文本：

```
This servlet is functional.
```

为了实现这一目标，新建一个项目，将其命名为 *Servlet*。在这个项目中，在包 `org.cookbook.ch11` 中，新建一个类 `ServletClass`，并把例 11-2 中的代码添加到其中。

为了确保导入成功，在编译路径中包含 *servlet-api.jar* 文件，该文件是 Tomcat 5.x 自带的（如果你使用的是 Tomcat 4.x，应包含 *servlet.jar* 文件）。按照路径 *jakarta-tomcat-5.0.19\common\lib\servlet-api.jar*，可以找到 *servlet-api.jar* 文件；在 Package Explorer 视图中右击 *Ch11_02* 项目，选择 Properties → Java Build Path → Add External JARs，并将 *servlet-api.jar* 添加到编译路径中。

在将 *servlet-api.jar* 添加到编译路径中之后，应该可以编译 servlet 的 *.class* 文件。选择 Project → Build Project，创建 *ServletClass.class*。

参考

11.5 节，在 Tomcat 中安装 Servlet；*Java Servlet Programming*（O'Reilly）一书的第 2 章；*JavaServer Page*（O'Reilly）一书的第 2 章；*Java Servlet and JSP Cookbook*（O'Reilly）一书的第 1 章。

11.5 在 Tomcat 中安装 Servlet

问题

你已经创建了一个 Servlet 的 *.class* 文件，并想在 Tomcat 中运行它。

解决方案

把这个 `.class` 文件添加到适当的 Tomcat 目录中（如 `webapps\Ch11\WEB-INF\class\org\cookbook\ch11`）。把 Servlet 的数据添加到 `web.xml` 文件中，重新启动 Tomcat，并导航到 Servlet 的 URL，如 `http://localhost:8080/ch11/org.cookbook.ch11.ServletClass`。

讨论

要安装 `.class` 文件，只需把它复制到 Tomcat 目录结构中的适当目录中即可。由于这个 Servlet 位于 `org.cookbook.ch11` 包中，而且 Tomcat 的目录结构必须镜像包结构，所以应把 `ServletClass.class` 文件添加到 `webapps\Ch11\WEB-INF\class\org\cookbook\ch11` 目录中：

```
webapps
|__ch11
|    |__WEB-INF
|        |__classes
|            |__org
|                |__cookbook
|                    |__ch11      Servlet 的代码放在此处
|__lib
```

为了在 Tomcat 中安装这个 Servlet，应创建部署描述符文件 `web.xml`。在这个 XML 文件中，使用两个元素，`<servlet>` 和 `<servlet-mapping>`，向 Tomcat 注册这个 Servlet。我们要使用的 `web.xml` 文件如例 11-3 所示。

例 11-3: web.xml 文件

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
    "http://java.sun.com/dtd/web-app_2_3.dtd">
<web-app>
  <display-name>Example Applications</display-name>

  <servlet>
    <servlet-name>Servlet</servlet-name>
    <servlet-class>org.cookbook.ch11.ServletClass</servlet-class>
  </servlet>

  <servlet-mapping>
    <servlet-name>Servlet</servlet-name>
    <url-pattern>/org.cookbook.ch11.ServletClass</url-pattern>
  </servlet-mapping>
</web-app>
```


右击 Servlet 项目，并选择 New → File，以创建这个新的 XML 文档。要在 Eclipse 中打开 `web.xml` 文件，可右击该文件，并选择 Open With → Text Editor（除非你已经在 Eclipse 中安装了一个专用的 XML 编辑器，作为一个插件，否则，Eclipse 将在默认的 XML 编辑器中打开 XML 文档，这个默认的 XML 编辑器可能是一个 Web 浏览器）。保存该文件，并把它复制到 `ch11` 目录下的 `WEB-INF` 目录中：

```
webapps
|__ch11
    |__WEB-INF                web.xml 文件保存在此
        |__classes
        |__lib
```

然后，（重新）启动 Tomcat，以便它把所有文件都复制到其工作目录中。现在，在浏览器中导航到 `http://localhost:8080/ch11/org.cookbook.ch11.ServletClass`。应该可以看到这个新的 Servlet 在运行，如图 11-4 所示。

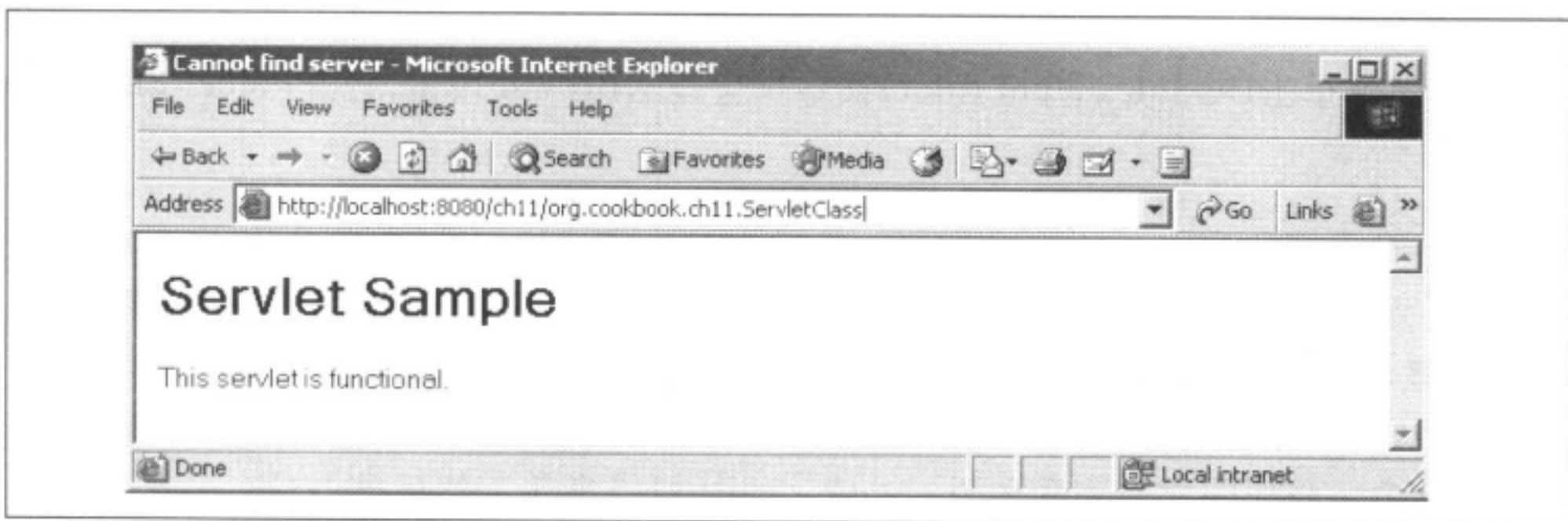


图 11-4：一个可运行的 Servlet

至此，我们已经使用 Eclipse 创建了可运行的、编译过的 Servlet 代码。但不可否认的一件令人头疼的事情是，必须把文件从 Eclipse 的目录复制到 Tomcat 的目录中。要了解如何在 Eclipse 中直接使用 Tomcat 目录中的文件，请参阅接下来的几节。

参考

11.6 节，就地创建 Servlet；11.7 节，就地编辑 `web.xml` 文件；*Java Servlet Programming*（O'Reilly）一书的第 2 章和第 3 章；*Java Servlet and JSP Cookbook*（O'Reilly）一书的第 2 章。

11.6 就地创建 Servlet

问题

你想就地开发 Web 应用程序，不必从 Eclipse 目录中复制文件。

解决方案

把项目的输出文件夹设置为 Web 容器的适当的目标目录。

讨论

作为一个例子，我们在此创建一个新的项目 *ServletInPlace*，并把所需的 Tomcat 文件存储在 Tomcat 目录中。通过打开 New Project 对话框，并输入项目名称 *ServletInPlace*，即可创建这个项目。要指定把编译输出存储在 Tomcat 目录中，而非存储在本地，可单击 Next，然后单击 Default output folder 文本框旁边的 Browse 按钮，打开 Folder Selection 对话框。

下一步，单击 Create new folder 按钮，然后在 New Folder 对话框中单击 Advanced 按钮。为了连接到要使用的 Tomcat 目录，请选中 Link to folder in the file system 复选框，并单击 Browse 按钮。找到用于存储编译代码的目标目录，*jakarta-tomcat-5.0.19\webapps\Ch11\WEB-INF\classes*，并单击 OK，再次打开 New Folder 对话框。在此对话框中，将新的文件夹命名为 *output*，如图 11-5 所示，然后单击 OK。

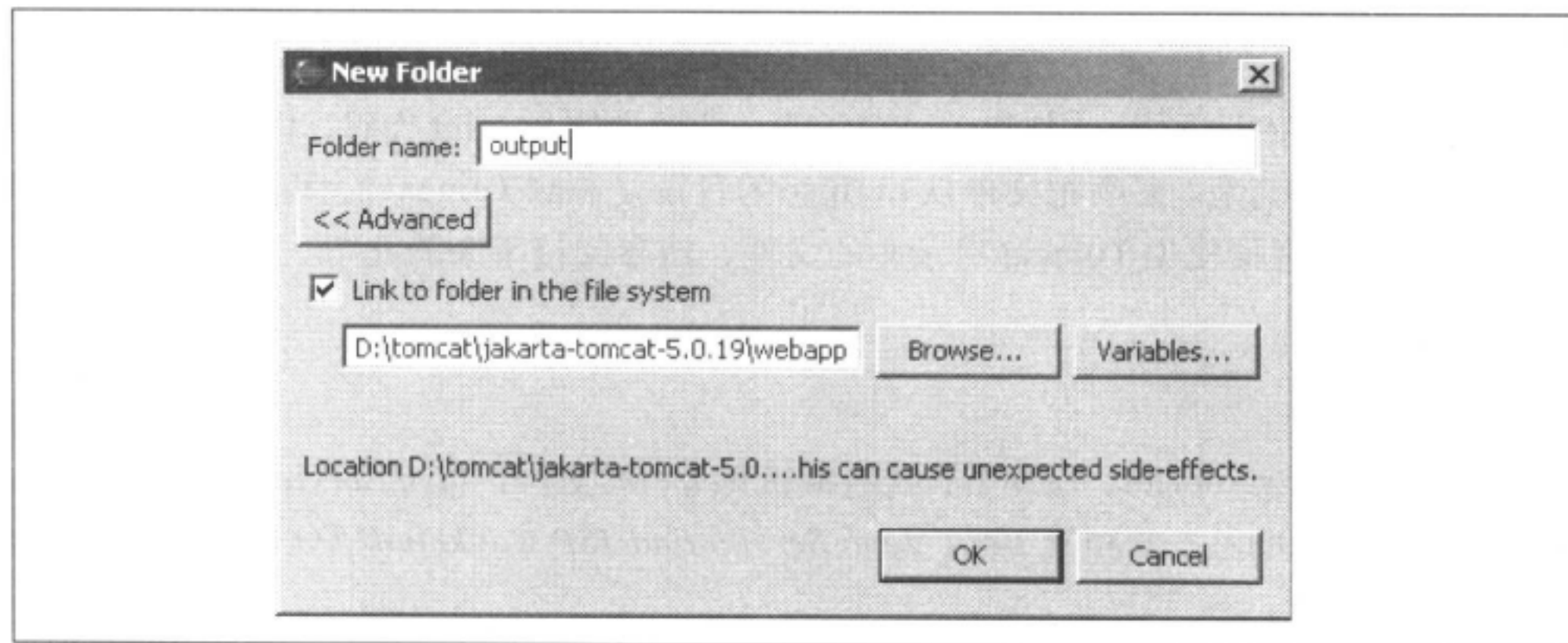


图 11-5：建立一个新的输出文件夹

当编译这个项目时，所有编译输出都将存储在 *classes* 文件夹内的指定位置（在本例中，实际上是存储在 *classes\org\cookbook\ch11* 文件夹中，后面接我们使用的包名称）。创建类 *ServletInPlaceClass*，并输入例 11-4 所示的代码。

例 11-4：就地创建 Servlet

```
package org.cookbook.ch11;

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class ServletInPlace extends HttpServlet
{
    public void doGet(HttpServletRequest request,
        HttpServletResponse response)
        throws IOException, ServletException
    {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();

        out.println("<HTML>");
        out.println("<HEAD>");
        out.println("<TITLE>");
        out.println("A Servlet Example");
        out.println("</TITLE>");
        out.println("</HEAD>");
        out.println("<BODY>");
        out.println("<H1>");
        out.println("Working With Servlets");
        out.println("</H1>");
        out.println("Developing servlets in place");
        out.println("</BODY>");
        out.println("</HTML>");
    }
}
```

与以前一样，把 *servlet-api.jar* 添加到编译路径中。现在，当编译这个项目时，*ServletInPlace.class* 自动存储在 Tomcat 的 *webapps\WEB-INF\classes\org\cookbook\ch11* 目录中。

参考

11.7 节，就地编辑 *web.xml* 文件；*Eclipse*（O'Reilly）一书的第 9 章。

11.7 就地编辑 web.xml 文件

问题

你想在 Eclipse 中编辑 *web.xml* 文件，而该文件仍然保存在 Tomcat 目录中。

解决方案

在项目中连接所需的文件，如 *web.xml* 文件。

讨论

为了在 Eclipse 中访问 *web.xml* 文件，可以使它成为一个连接文件，即使它位于 *webapps\ch11\WEB-INF* 目录中也无妨。为此，右击 *ServletInPlace* 项目，选择 New → File，单击 Advanced 按钮，选中 Link to file in the file system 复选框，并单击 Browse 按钮。浏览到 *webapps\ch11\WEB-INF* 目录，并单击 Open。然后在打开的 New File 对话框中，输入要连接的文件名称，*web.xml*，并单击 OK。这样就可以把 *web.xml* 文件添加到项目中，如图 11-6 中左侧的 Package Explorer 视图所示。

可以对 *web.xml* 文件进行编辑，以支持 *ServletInPlace Servlet*，如例 11-5 所示。

例 11-5: web.xml 文件的新版本

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.
//DTD Web Application 2.3//EN" "http://java.sun.com/dtd/web-app_2_3.dtd">
<web-app>
  <display-name>Example Applications</display-name>

  <servlet>
    <servlet-name>Servlet</servlet-name>
    <servlet-class>org.cookbook.ch11.ServletClass</servlet-class>
  </servlet>

  <servlet>
    <servlet-name>ServletInPlace</servlet-name>
    <servlet-class>org.cookbook.ch11.ServletInPlaceClass</servlet-class>
  </servlet>

  <servlet-mapping>
    <servlet-name>Servlet</servlet-name>
    <url-pattern>/org.cookbook.ch11.ServletClass</url-pattern>
  </servlet-mapping>

  <servlet-mapping>
    <servlet-name>ServletInPlace</servlet-name>
```



```
<url-pattern>/org.cookbook.ch11.ServletInPlaceClass</url-pattern>
</servlet-mapping>

</web-app>
```

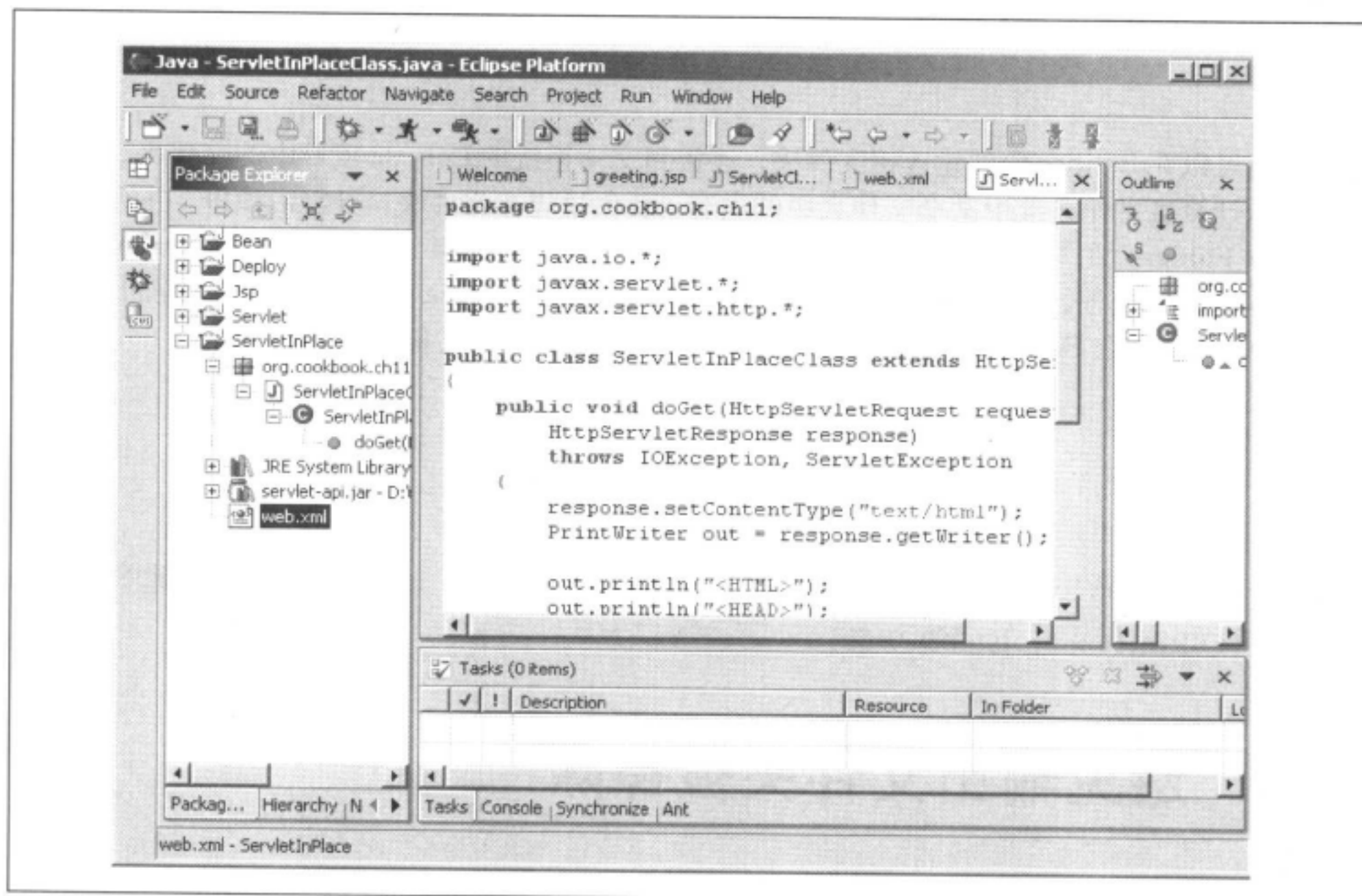


图 11-6：连接到一个文件

在对 `web.xml` 文件进行了上述编辑之后，重新启动 Tomcat。我们已经编译过了这个 Servlet 的代码，所以可直接导航到新版 Servlet 的 URL，`http://localhost:8080/ch11/org.cookbook.ch11.ServletInPlaceClass`，如图 11-7 所示。

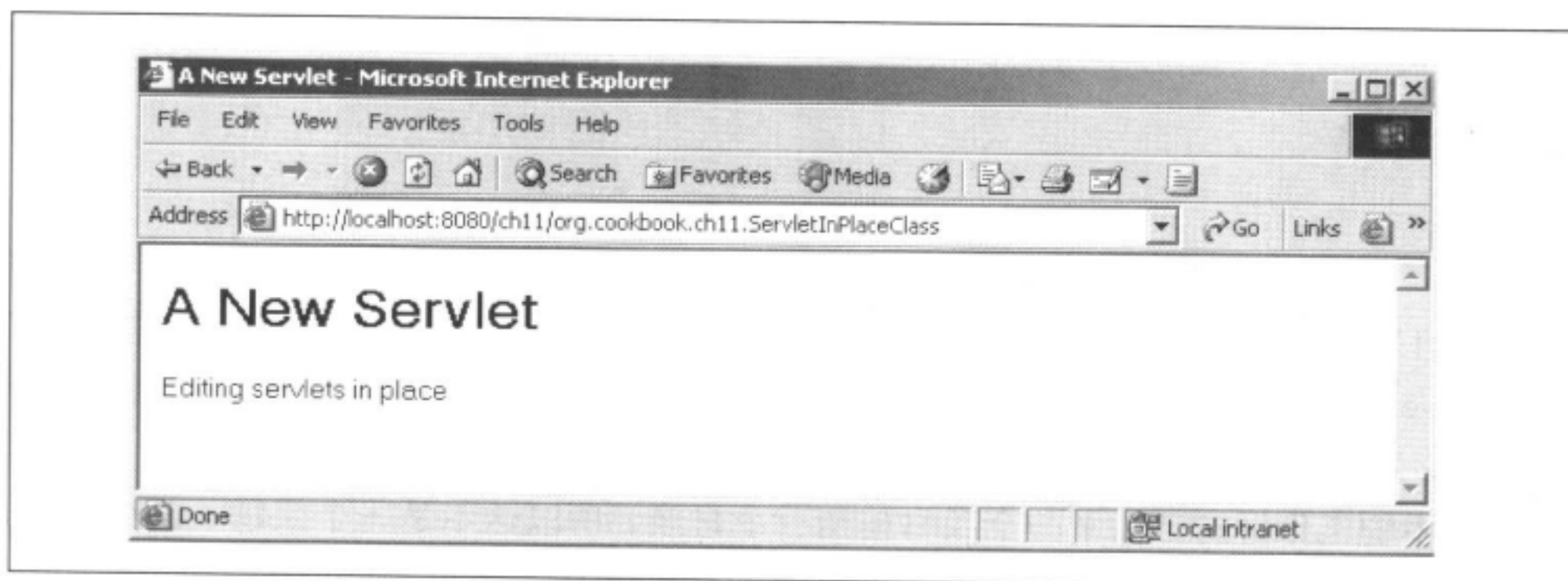


图 11-7：运行就地开发的 Servlet

如你所见，让 Eclipse 来处理存储编译文件和就地编辑导入文件的细节，可以大大简化开发工作，再也不必先修改文件，然后再把它们复制到 Web 容器的目录中。如果你正在考虑使用 JSP/Servlet 进行严肃的 Web 开发，这是一条正确的道路；整个开发/测试/修订周期将因此而大大简化。

注意：还可以在 Eclipse 中创建连接文件夹。连接到文件系统的 Eclipse 项目中的文件夹，可以显示计算机文件系统中某个文件夹中的内容。要创建连接文件夹，可右击项目，并选择 New → Folder。在 New Folder 对话框中单击 Advanced 按钮，并选中 Link to folder in the file system 复选框；然后单击 Browse 按钮，浏览到要在 Eclipse 中显示的文件系统的文件夹。单击 OK，在 Folder Name 文本框中输入这个新文件夹的名称，并单击 OK。

参考

11.4 节，创建 Servlet；11.5 节，在 Tomcat 中安装 Servlet；11.6 节，就地创建 Servlet；*Eclipse* (O'Reilly) 一书的第 9 章。

11.8 避免输出文件夹被擦除问题

现在你正在使用 Web 服务器目录中的文件，想改变 Eclipse 的默认擦除（即删除）行为——擦除编译目录中的所有文件，因为这样做会删除这些目录中还需要使用的文件。

解决方案

要阻止 Eclipse 在每次编译项目时擦除项目的输出目录，可选择 Project → Properties → Java Compiler → Use Project Settings → Build Path，并取消 Scrub output folders on full build 复选框。

讨论

当你开始使用 Web 容器目录中的文件，而不仅仅是 Eclipse 本地工作区时，在进行编译时 Eclipse 将删除输出目录中的所有文件（这是它的默认行为），从而产生一个问题。例如，这些文件可能是尚需要使用的其他 Servlet，或者是服务器需要的配置文件。在使用 Eclipse 中的工作区时，由于一个项目使用一个目录，所以这不是一个问题；但当目录，就像 Web 容器中的目录，由许多文件共享时，这可能是一个严重的问题。

要改变Eclipse的默认行为,可选择Project → Properties → Java Compiler → Use Project Settings → Build Path,并取消Scrub output folders on full build复选框,如图11-8所示。这将阻止Eclipse在默认情况下擦除输出文件夹。

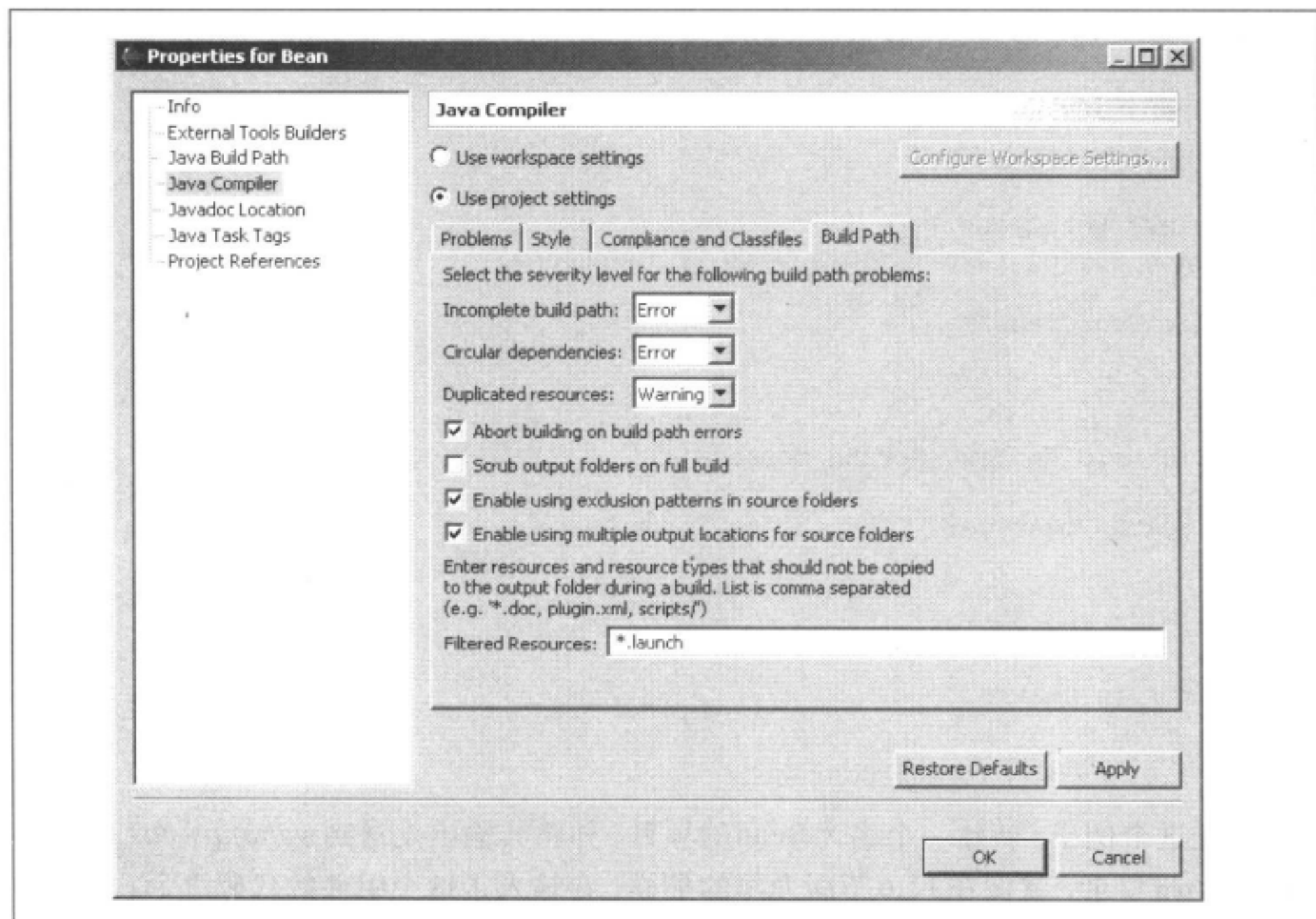


图 11-8: 避免输出文件夹被擦除

11.9 连接到 Java 组件

问题

你想在 Eclipse 中开发 Java 组件,并将它们连接至 JSP。

解决方案

就地开发 Java 组件代码和 JSP。为此,只需将项目的输出文件夹设置为保存组件代码的 `WEB-INF\classes` 目录,并在 Eclipse 中创建将该组件用作连接文件的 JSP 即可。

讨论

可以将Java代码编译成Java组件，并从JSP中访问该代码。既然你已经知道了如何使用连接文件，那么使用JavaBeans进行开发应该没有问题。作为一个例子，请看例11-6。这个组件设置了名为text的属性，使之保存字符串“This bean is functional.”。

例 11-6：一个 Java 组件

```
package org.cookbook.ch11;

public class BeanClass {
    private String text = "This bean is functional.";

    public BeanClass()
    {
    }

    public void setText(String message)
    {
        text = message;
    }

    public String getText()
    {
        return text;
    }
}
```

为了建立这个例子，新建一个名为Bean的项目，并将其输出发送到webapps\ch11\WEB-INF\classes目录，就像在11.6节所介绍的那样。在输入了这个组件的代码之后，编译项目，创建并安装BeanClass.class。

连接到这个组件的JSP文件如例11-7所示。在这个例之中，使用JSP的<jsp:useBean>元素创建一个JavaBean对象，然后使用<jsp:getProperty>元素获得一个组件属性的值。

例 11-7：连接至一个 Java 组件

```
<HTML>
  <HEAD>
    <TITLE>Setting a Property Value</TITLE>
  </HEAD>

  <BODY>
    <H1>Setting a Property Value</H1>

    <jsp:useBean id="bean1" class="org.cookbook.ch11.BeanClass" />

    Got this from the bean: <jsp:getProperty name="bean1" property="text" />
  </BODY>
</HTML>
```

这个JSP文件显示了msg属性中原来的消息及其被设置后的新值。将这个新建的JSP文件作为一个连接文件，保存在webapps\Ch11目录中，并导航到http://localhost:8080/ch11/Bean.jsp，如图11-9所示。

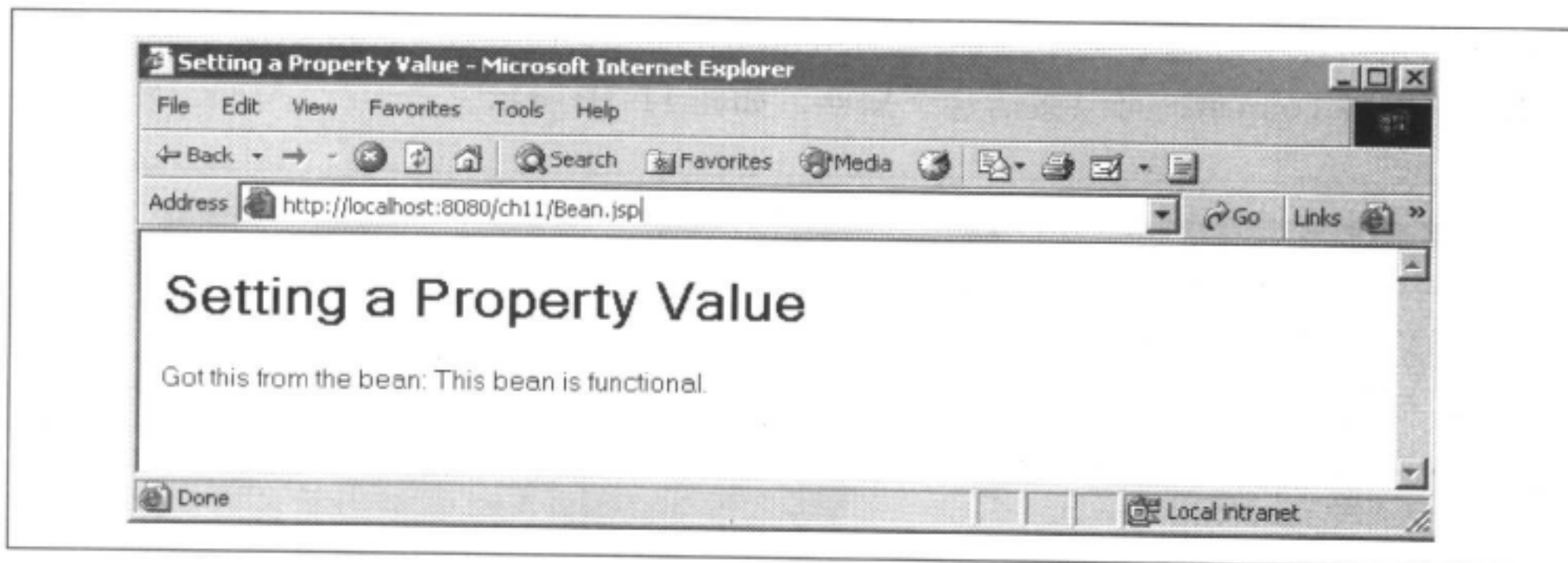


图 11-9：连接到一个 Java 组件

正如图所示，JSP 文件确实连接到了这个 Java 组件。使用输出文件夹和连接文件，以这种方式开发应用程序不存在任何问题。

参考

11.3 节，创建 JSP 文件；*JavaServer Pages* (O'Reilly) 一书的第 6 章；*Java Servlet and JSP Cookbook* (O'Reilly) 一书的第 7 章；*Eclipse* (O'Reilly) 一书的第 9 章。

11.10 使用 Tomcat 插件

问题

你想自动执行某些 Tomcat 过程，如启动和停止 Tomcat 以及调试 Tomcat 项目。

解决方案

使用一个 Tomcat 插件，如 Sysdeo 插件。从站点 <http://www.sysdeo.com/eclipse/tomcatPlugin.html> 下载这个插件，并将其解压到 Eclipse 的 *plugins* 目录中。这将在 Eclipse 工具栏上添加几个按钮，用于启动和停止 Tomcat。

讨论

通过 Sysdeo 插件可以从 Eclipse 内部启动和停止 Tomcat，而且可以从站点 <http://www.sysdeo.com/eclipse/tomcatPlugin.html> 免费下载这个插件。通过解压到 Tomcat 的 `plugins` 目录可以安装这个插件，然后选择 Window → Customize Perspective，展开 Other 节点，并单击 Tomcat，可以激活这个插件，如图 11-10 所示。

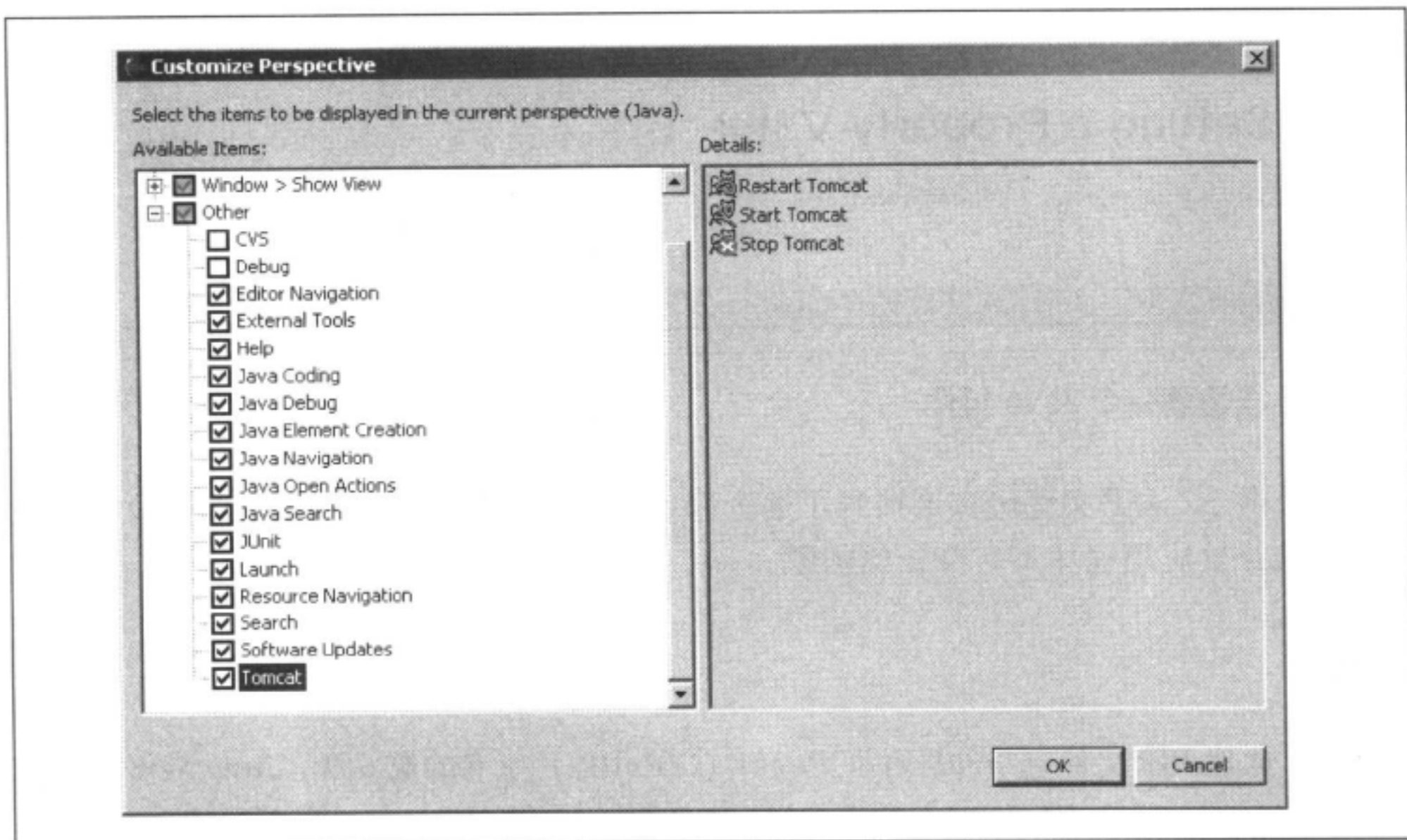


图 11-10：配置 Tomcat 插件

注意：要进一步了解关于安装、使用 and 创建插件的内容，请参阅后面两章。

如图 11-11 所示，这将在 Eclipse 中新增一个 Tomcat 菜单，并在 Eclipse 工具栏上增加 3 个 Tomcat 按钮（可以看到，新增的按钮在 Navigate 菜单下方），分别用于启动、停止和重启 Tomcat。

还需要将这个插件连接到你的 Tomcat 安装。要进行连接，可选择 Window → Preferences，展开 Tomcat 节点。单击 Tomcat home 文本框旁边的 Browse 按钮，浏览到你的 Tomcat 安装，如图 11-12 所示。

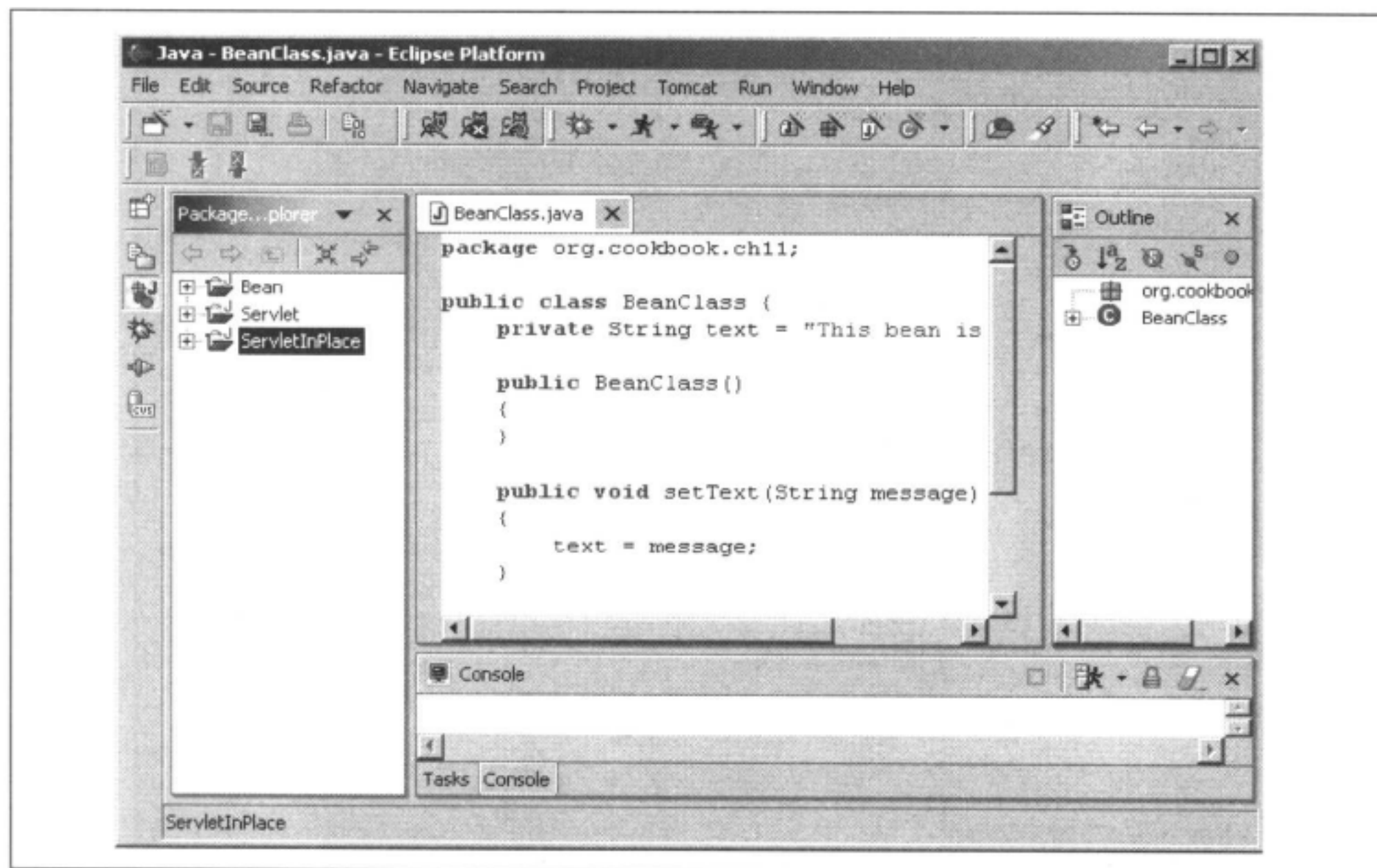


图 11-11: Tomcat 菜单和工具栏上的按钮

当打算创建一个 Tomcat 项目时，这个插件可以处理将文件存储在 Tomcat 安装中的细节问题。只需选择 `File → New → Project`，并选择 `Tomcat Project` 作为项目类型即可，如图 11-13 所示。

如图 11-14 所示，在正确的 Tomcat 目录中创建了项目文件，这些文件将出现在 Package Explorer 视图中。

通过 Sysdeo 插件，还可以在 Tomcat 项目运行时使用 Eclipse 调试器对项目进行调试。对于 Web 开发来说，这是非常有用的，否则，能够跟踪错误的全部消息就是 Tomcat 在浏览器中显示的含糊的“Error 500 Server Error”消息。

参考

Eclipse (O'Reilly) 一书的第 9 章。

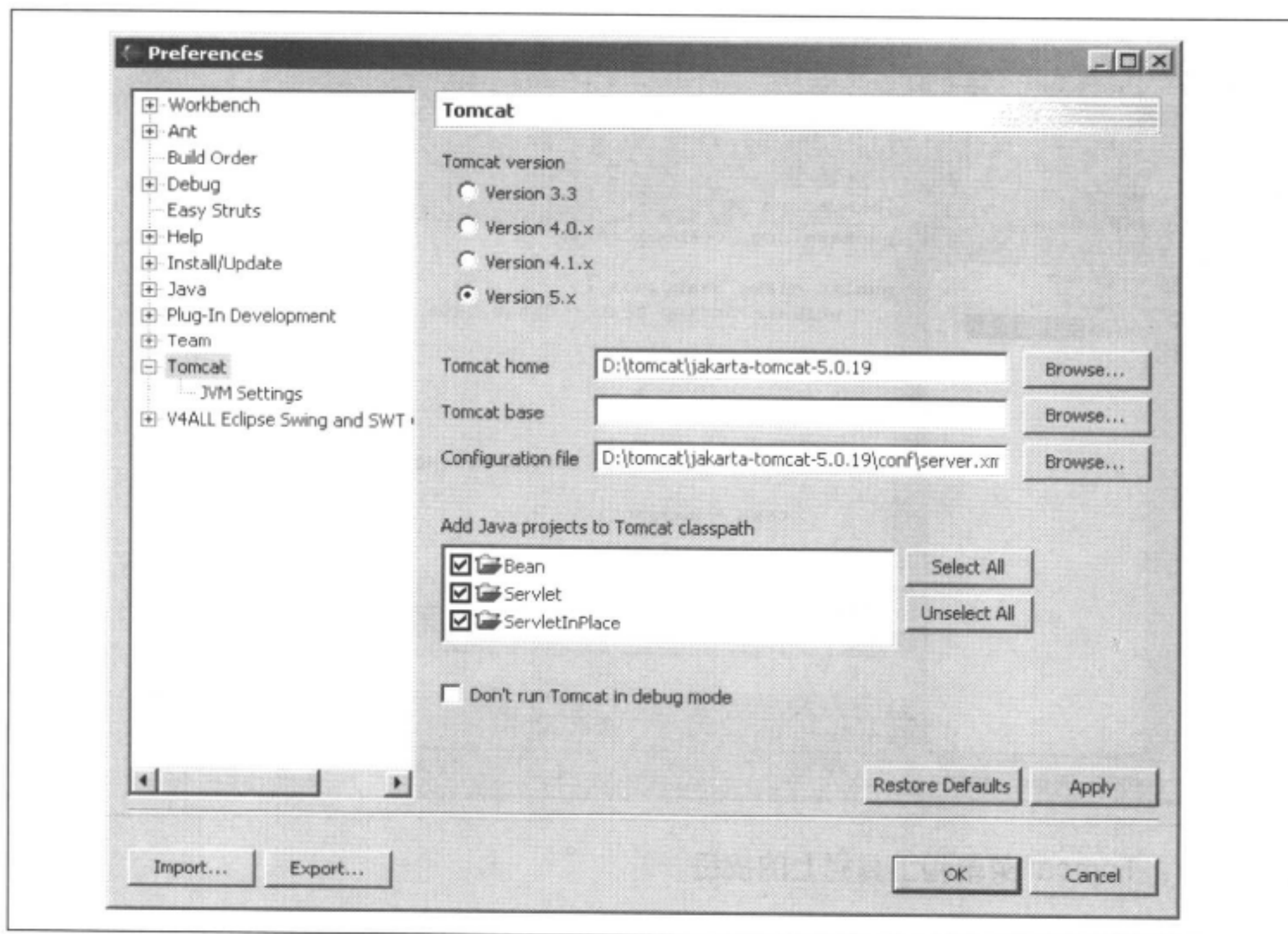


图 11-12: 配置 Tomcat 插件 Sysdeo

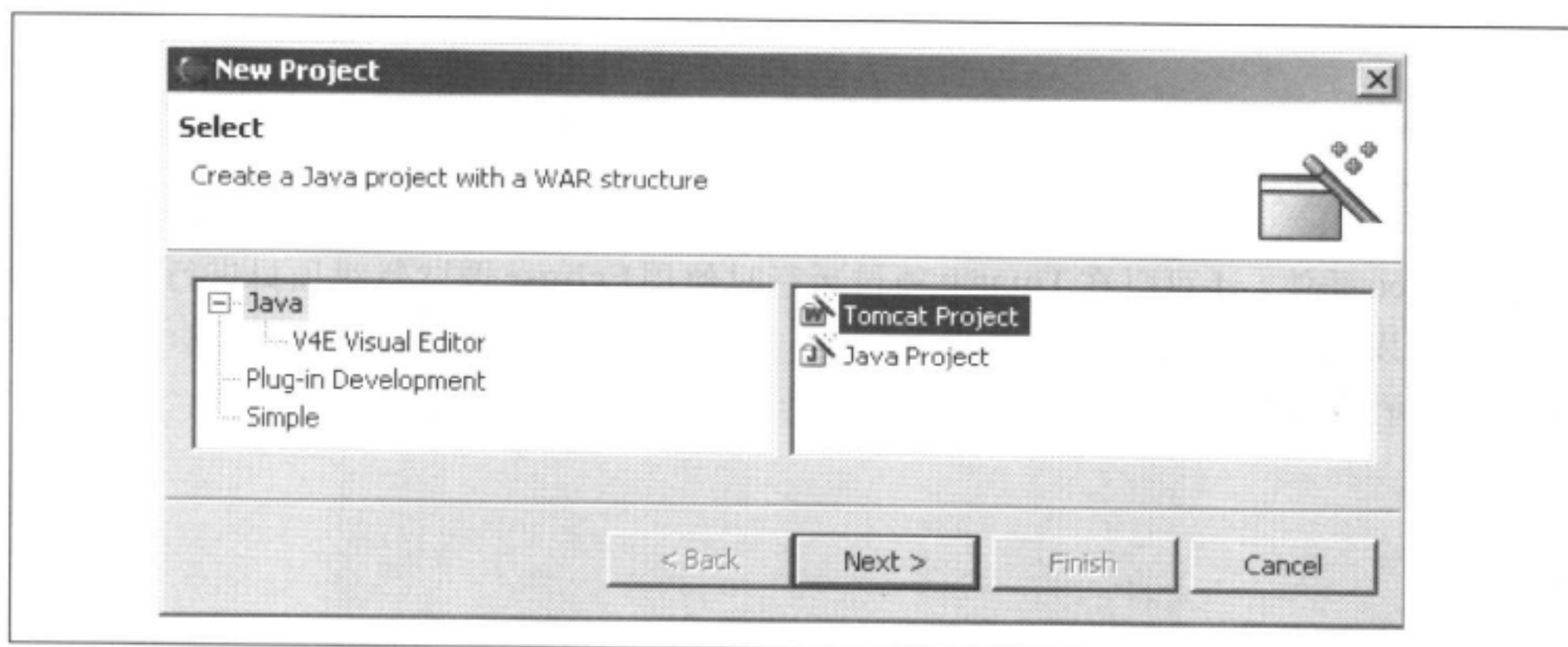


图 11-13: 创建一个 Tomcat 项目

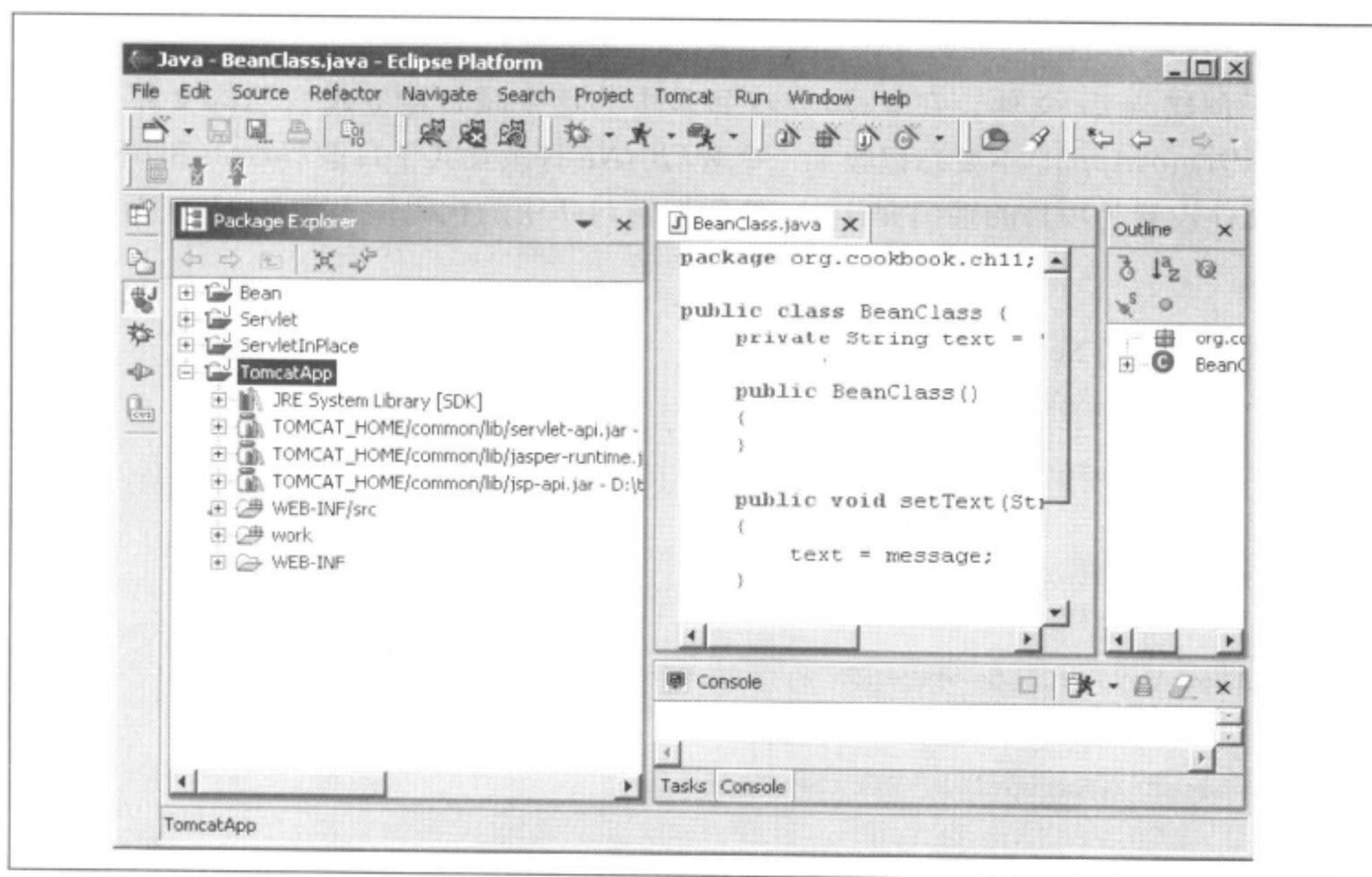


图 11-14：一个新建的 Tomcat 项目

11.11 创建 WAR 文件问题

你想为一个 Web 应用程序创建部署文件。

解决方案

把 Web 应用程序压缩成一个 .war 文件，这就是用来部署 Web 应用程序的文件类型。可以从 Eclipse 内部使用 Ant 创建 .war 文件。

讨论

为了说明如何使用 Eclipse 部署 Web 应用程序，我们将部署例 11-8 中的示例 Servlet，即项目 *DeployApp*，可以从本书的 O'Reilly 站点找到该项目。在将应用程序压缩成一个 .war 文件之后，可以将这个 .war 文件拖到 Tomcat 的 *webapps* 目录中。当重启 Tomcat 时，Tomcat 将解压并安装这个文件，即部署这个应用程序。

创建要部署的应用程序

为了便于创建 *.war* 文件，在 *webapps* 目录中为这个项目建立起自己的文件夹（即 *webapps\DeployApp*）。不要忘记添加一个 *WEB-INF* 目录及其子目录 *classes* 和 *lib*，并把 *classes* 文件设置为项目的输出文件夹。然后把例 11-8 中的代码输入到 *DeployClass.java* 文件中。

例 11-8：要部署的 Servlet

```
package org.cookbook.ch11;

import javax.servlet.http.HttpServlet;

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class DeployClass extends HttpServlet {
    public void doGet(HttpServletRequest request,
        HttpServletResponse response)
        throws IOException, ServletException
    {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();

        out.println("<HTML>");
        out.println("<HEAD>");
        out.println("<TITLE>");
        out.println("Deploying a Project");
        out.println("</TITLE>");
        out.println("</HEAD>");
        out.println("<BODY>");
        out.println("<H1>");
        out.println("Deploying a Project");
        out.println("</H1>");
        out.println("This deployed project is functional.");
        out.println("</BODY>");
        out.println("</HTML>");
    }
}
```

用于这个项目的 *web.xml* 文件如例 11-9 所示；把这个文件添加到 *WEB-INF* 文件夹中。然后编译项目，（重新）启动 Tomcat，并导航到 *http://localhost:8080/DeployApp/org.cookbook.ch11.DeployClass*，以确认这个 Servlet 是可以运行的。这就是我们要部署的 Servlet。

例 11-9：用于要部署的 Web 应用程序的 *web.xml* 文件

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
"http://java.sun.com/dtd/web-app_2_3.dtd">
```



```
<web-app>
  <display-name>Example Applications</display-name>

  <servlet>
    <servlet-name>DeployApp</servlet-name>
    <servlet-class>org.cookbook.ch11.DeployClass</servlet-class>
  </servlet>

  <servlet-mapping>
    <servlet-name>DeployApp</servlet-name>
    <url-pattern>/org.cookbook.ch11.DeployClass</url-pattern>
  </servlet-mapping>
</web-app>
```

创建要部署的 .war 文件

我们已经编译了要部署的 Servlet，下一步任务是将其打包成一个可部署的 .war 文件。在 Eclipse 中，打包的最佳方式是使用 Ant，并使用例 11-10 所示的 *build.xml* 文件。这个 *build.xml* 文件接收 *DeployApp* 项目的输出，并将其压缩成一个名为 *DeployAppWar.war* 的 .war 文件，存储在 *webapps\DeployApp* 目录中。

例 11-10: build.xml for the servlet to deploy

```
<?xml version="1.0" encoding="UTF-8" ?>
<project name="DeployApp" default="Main Build" basedir=".">
  <property name="project" location="d:/tomcat/jakarta-tomcat-5.0.19/webapps/DeployApp" />
  <property name="wardir" location="d:/tomcat/jakarta-tomcat-5.0.19/webapps/DeployApp" />
  <property name="warfile" location="${wardir}/DeployAppWar.war" />

  <target name="Initialize">
    <delete file="${warfile}" />
  </target>
  <target name="War">
    <jar destfile="${warfile}" basedir="${project}" />
  </target>
  <target name="Main Build" depends="Initialize, War">
    <echo message="Building the .war file" />
  </target>
</project>
```

将 *build.xml* 文件添加到项目中。在编译项目之后，右击 *build.xml* 文件，选择 Run Ant，并单击 Run，以创建 *DeployAppWar.war* 文件。

部署 .war 文件

在创建了 *DeployAppWar.war* 文件之后，只需将该文件添加到目标机器上的 Tomcat *webapps* 目录中，并（重新）启动 Tomcat，即可部署应用程序。Tomcat 使用这个 .war

文件的名称，解压这个文件，并将这个 Web 应用程序安装到一个名为 *DeployAppWar* 的目录中。

在把 *DeployAppWar.war* 文件拖到 *webapps* 目录中之后，（重新）启动 Tomcat，并导航到 <http://localhost:8080/DeployAppWar/org.cookbook.ch11.DeployClass>，以检查部署是否成功。应该可以看到如图 11-15 所示的结果。以后，你应该使用 Eclipse 为 Web 应用程序创建了部署包。

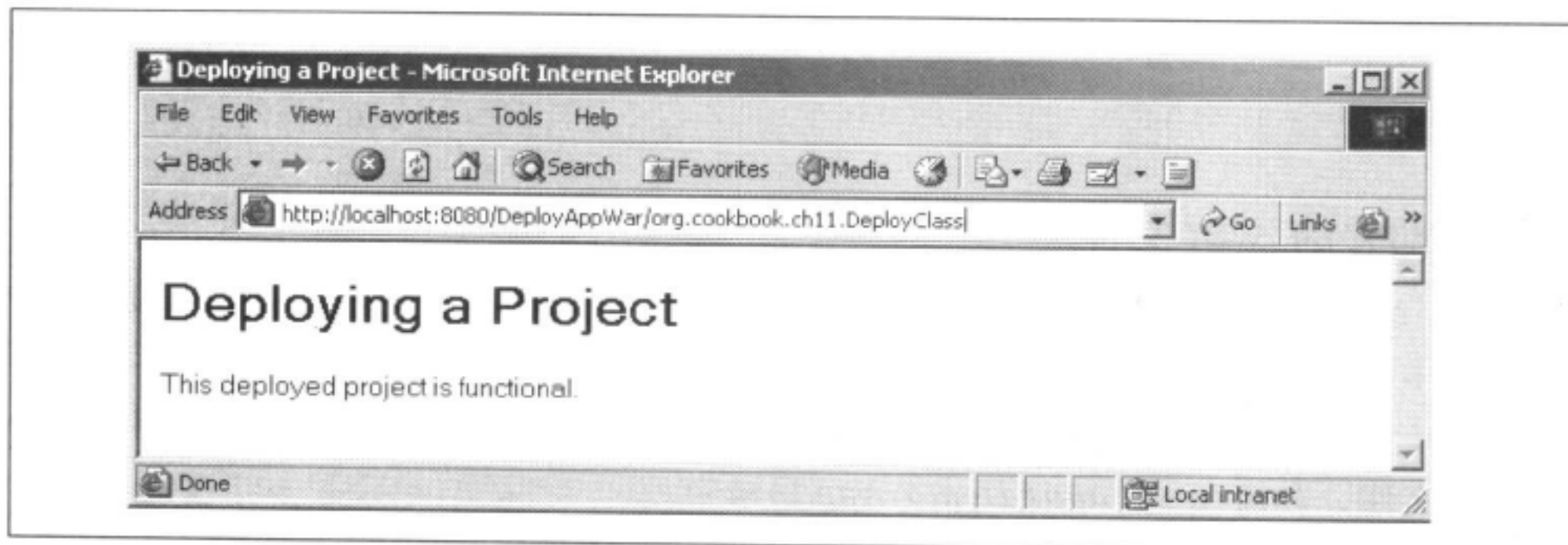


图 11-15：一个已部署的项目

注意：也可以使用 Ant 直接部署 *.war* 文件，即使用 Ant 任务（如 *ftp*）将 *.war* 文件发送到远程机器上需要使用的 Web 服务器目录中。

参考

7.1 节，将 Ant 连接至 Eclipse；*Tomcat: the Definitive Guide* (O'Reilly) 一书的第 3 章；*Java Servlet and JSP Cookbook* (O'Reilly) 一书的第 2 章。

创建插件：扩展点、 动作和菜单

12.0 简介

从本章开始，我们将介绍如何创建 Eclipse 插件。也就是说，我们要扩展 Eclipse 及其功能。迄今为止，我们一直在使用现成的 Eclipse；从现在起，我们将增加自己的代码，以扩展这个开发环境。

能够创建自己的插件是 Eclipse 最强大的功能之一，尽管这种工作并非适合每一个人，但对此有所了解还是有益的。如你所料，Eclipse 插件是用 Java 编写的。

在这一章中，首先通过创建一个能够在 Eclipse IDE 中添加菜单和菜单项的插件，来介绍插件开发和插件开发环境（Plug-in Development Environment, PDE）。当用户选择这些自定义菜单项时，插件代码将做出响应。通过从头创建一个插件，来了解有关插件开发的全过程。

12.1 安装插件

问题

你想安装一个 Eclipse 插件。

解决方案

下载插件，并将其解压到 Eclipse 的 *plugins* 目录中。

讨论

已经有大量的插件可以使用，其中许多可以免费下载。要安装插件，应关闭 Eclipse（如果它正在运行的话），并把插件下载到 *eclipse* 目录中，这个目录中包含 *workspace* 和 *plugins* 目录。

注意：从站点 <http://www.eclipse-plugins.2y.net/eclipse/> 可以找到 450 多种 Eclipse 插件，其中大部分是免费的。每天从这个站点下载的插件大约有 7 000 个。其中许多插件弥补了 Eclipse 的不足，特别是针对 Swing、Struts 和 SWT 等环境的拖放开发。

下载的 Eclipse 插件都是压缩的，通常需要将其解压到 *eclipse* 目录中。解压时，插件的有关文件将自动存储在 *plugins* 和 *features* 目录中。

每一个插件在 Eclipse 的 *plugins* 目录中都有其自己的文件夹。通常，在每一个插件的文件夹中都可以找到下列文件：

**.jar*

插件的代码，存储在一个 *.jar* 文件中。

about.html

当用户请求关于插件的信息时显示该文件。

icons

存储图标的目录（标准格式为 GIF）。

lib

储存库的 *.jar* 文件。

plugin.xml

插件的信息清单，用于向 Eclipse 描述插件是什么。

plugin.properties

存储 *plugin.xml* 使用的文本数据。

注意：尽管通常把插件解压到 *eclipse* 目录中，但有些插件被设计成解压到 *plugins* 目录中。如果没有插件的安装说明，可使用解压工具打开插件，并查看它是如何展开的——文件 *plugin.xml* 总是必须解压到 *plugins* 目录的子目录中。

这些就是安装一个插件时所需要做的全部工作。在展开了插件的压缩文件之后，再次启动 Eclipse。可能会看到一个对话框，表示配置更新是在进行之中；必要时重启 Eclipse。

12.2 创建 plugin.xml 文件

问题

你想以最省事的方式创建一个新的插件。

解决方案

至少，每一个插件都需要一个 *plugin.xml* 文件，用于向 Eclipse 描述插件本身。如果你有了一个 *plugin.xml* 文件，就等于有了一个插件——一个甚至没有任何代码的插件。*plugin.xml* 文件的开头是一个标准的 XML 声明，添加一个 `<plugin>` 元素，并设置该元素的 `id`、`name`、`version` 和 `provider-name` 属性。然后，把 *plugin.xml* 文件存储在 *plugins* 目录中用来安装这个尚待开发的插件的文件夹中。

讨论

为了掌握在 Eclipse 中如何查看插件，我们将创建一个最简单的插件，它仅包含一个 *plugin.xml* 文件。这个文件的名称为 *plug-in manifest*，用于向 Eclipse 描述关于该插件的一切以及在何处可以找到 Java 支持代码（如果有的话）。因为对 Eclipse 而言，创建一个插件所需的全部信息就是一个插件信息清单，所以我们现在要创建一个插件信息清单，作为示例。

使用可保存纯文本的任何文本编辑器，包括 Eclipse，为插件 `org.cookbook.simple` 创建一个插件信息清单。下面是一个 *plugin.xml* 文件示例，其中指定了插件的名称、ID、版本号和供应商的名称：

```
<?xml version="1.0" encoding="UTF-8"?>
<plugin
  id="org.cookbook.simple"
  name="Simple Plug-in"
  version="1.0.0"
  provider-name="Plug-in Power Corp.">
</plugin>
```

插件被存储在 *eclipse/plugins* 文件夹下，能够反映插件的名称和版本号的文件夹中。为了说明我们创建的是这个简单插件的 1.0.0 版，把 *plugin.xml* 文件存储为 *eclipse/plugins/org.cookbook.simple_1.0.0/plugin.xml*，然后重启 Eclipse。

这个新的插件不具有任何功能，只是用来确认 Eclipse 把它作为一个真正的插件来对待，并查看插件注册表——选择 `Help → About Eclipse Platform`，并单击 `Plug-in Details` 按

钮，可以打开插件注册表。在插件注册表中可以找到这个新建的、简单的插件，如图 12-1 底部所示。

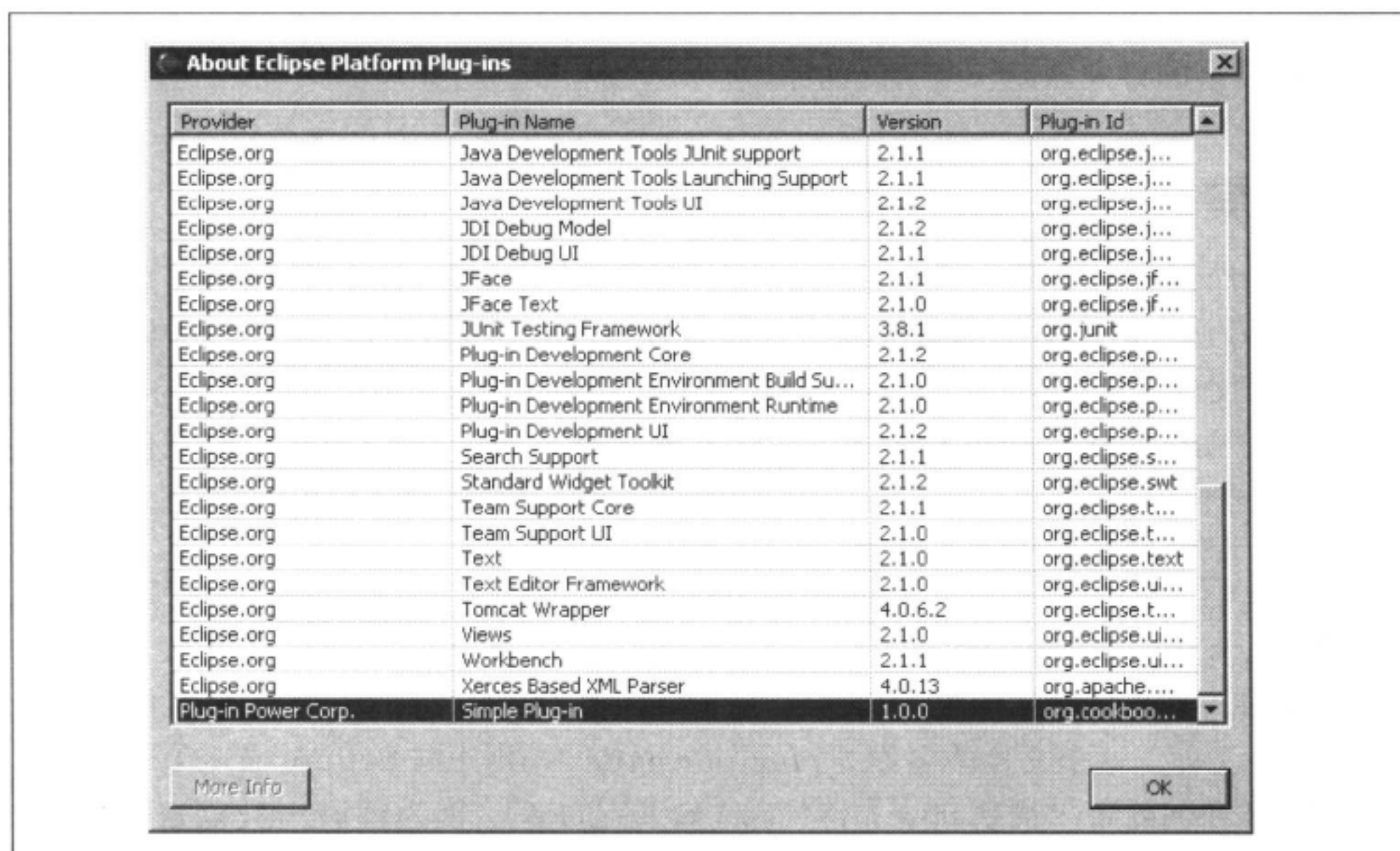


图 12-1：插件注册表中的简单插件

这个例子简单说明了如何使用创建信息清单。要开始创建插件的代码，请参阅接下来的几节。

参考

12.3 节，使用向导创建基于菜单的插件；*Eclipse* (O'Reilly) 一书的第 11 章和第 12 章。

12.3 使用向导创建基于菜单的插件

问题

你想创建一个能够在 Eclipse IDE 中添加菜单和菜单项的插件。

解决方案

使用 PDE 中提供的一种插件模板创建支持菜单的插件。

讨论

使用 Eclipse PDE，创建并修改自己的插件是一件相当简单的事情。下面是使用 PDE 能够创建的项目的类型：

Plug-in Project

标准的插件项目。

Fragment Project

能够用作插件的附件的项目。

Feature Project

包含一个或多个插件的项目。

Update Site Project

能够自动更新特性的 Web 项目。

作为一个例子，我们将在一个名为 *MenuPlugIn* 项目中创建一个新的插件，用于在 Eclipse IDE 中添加一个菜单和一个工具栏按钮。要创建这个插件项目，可选择 **File** → **New** → **Project**。在 **New Project** 对话框中左侧的列表框中，选择 **Plug-in Development**，并在右侧的列表框中选择 **Plug-in Project**，如图 12-2 所示。然后单击 **Next**。

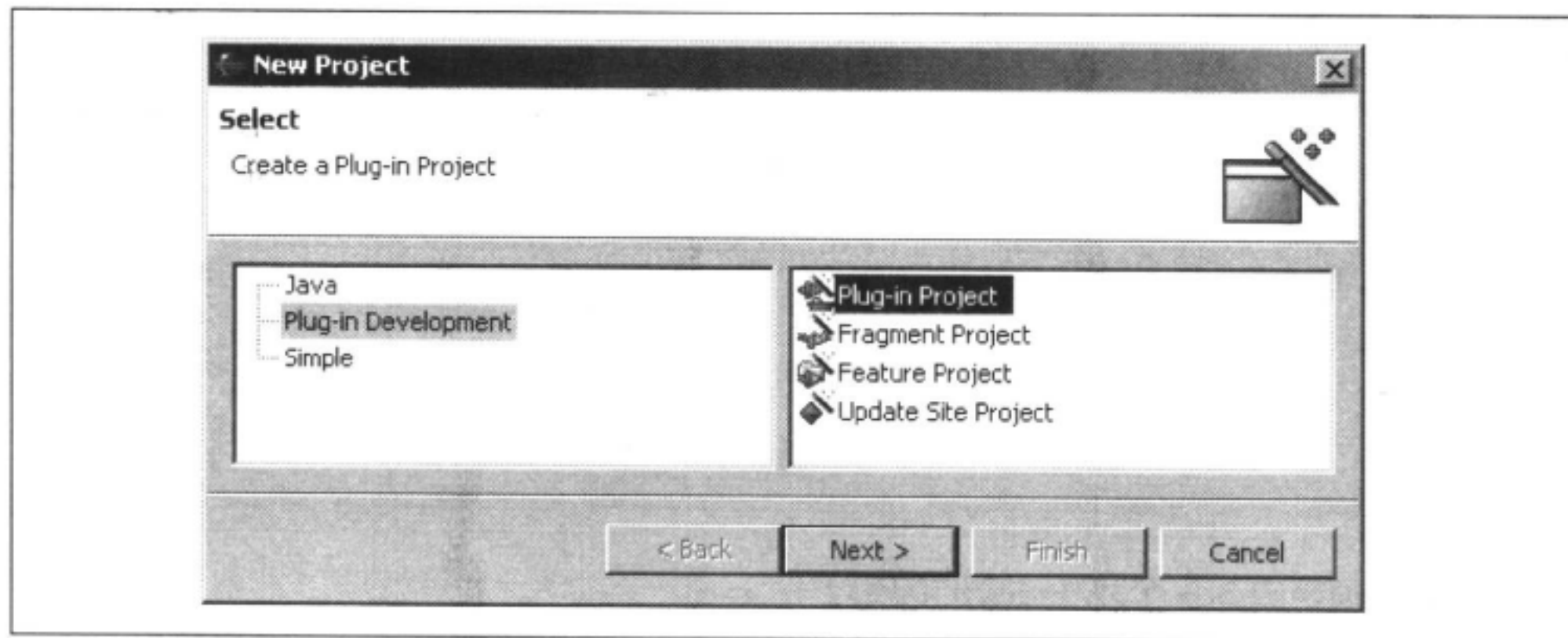


图 12-2：新建一个插件项目

在对话框的下一页，要求提供插件项目的名称。将这个项目名称命名为 `org.cookbook.ch12.MenuPlugIn`（这也是插件在 Eclipse 中使用时的 ID），如图 12-3 所示。单击 **Next**。

在对话框的下一页，接收默认的设置，如图 12-4 所示，并单击 **Next**，使这个插件项目成为一个基于 Java 的项目。

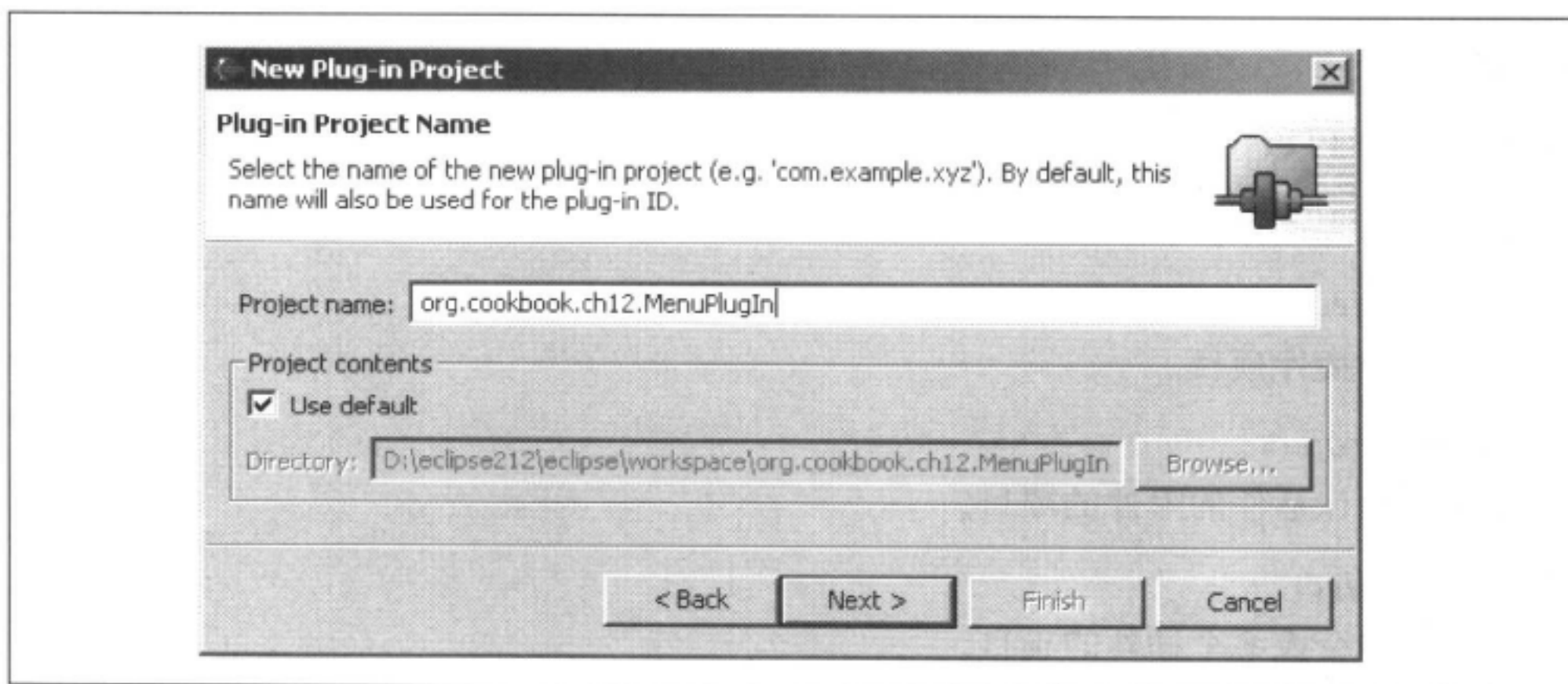


图 12-3：为插件命名

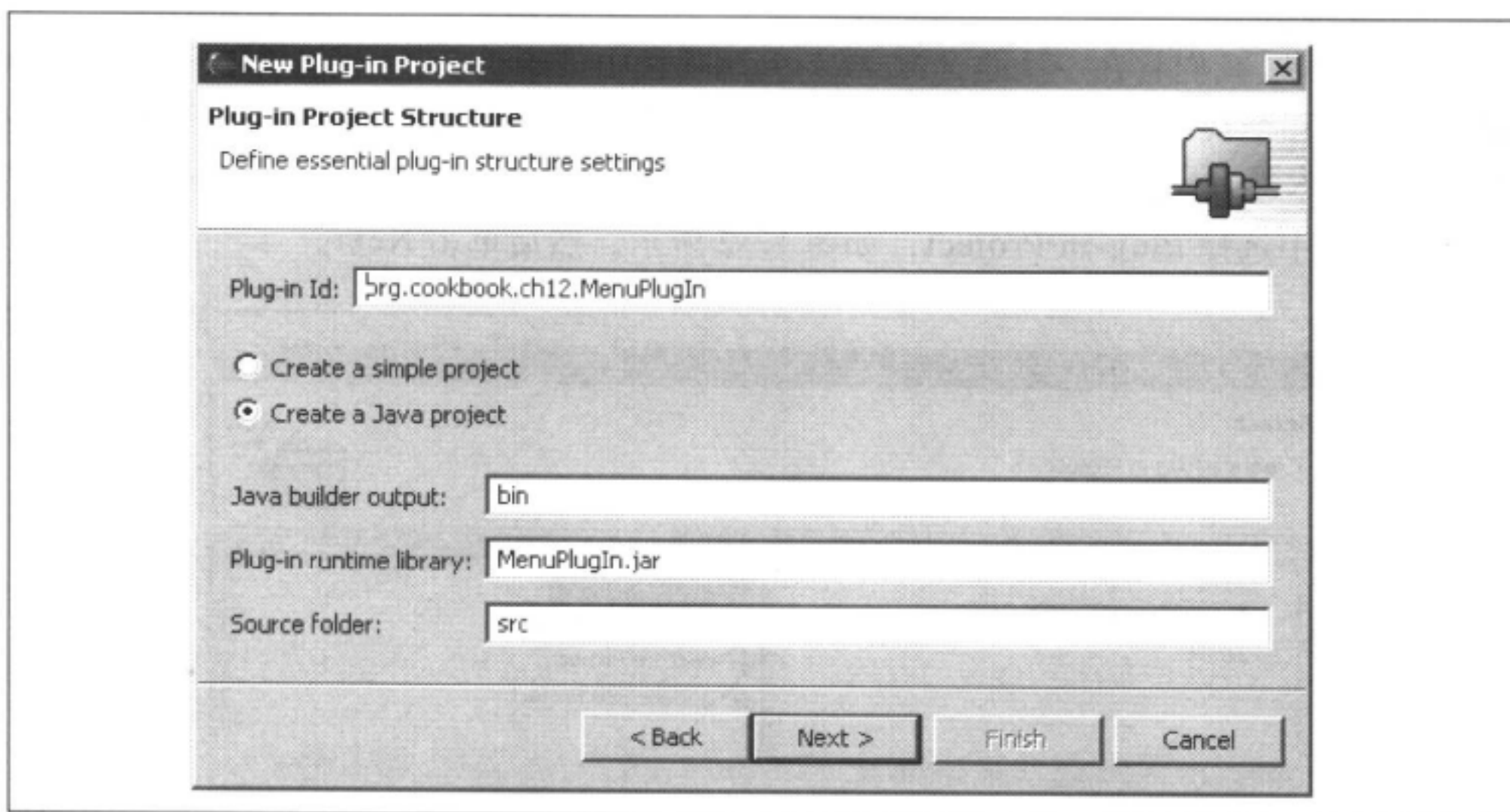


图 12-4：使项目成为一个基于 Java 的项目

在对话框的下一页，选择用于创建项目的向导。在本例中，单击左边的列表框中的“Hello, World”向导，如图 12-5 所示，并单击 Next。

在对话框的下一页，对插件进行配置。对于供应商的名称，输入“Eclipse Cookbook”，接收其他的默认设置，如图 12-6 所示。

单击 Next，输入当插件的菜单项或工具栏按钮被选中时显示的文本：“This plug-in is functional.”，如图 12-7 所示。单击 Finish，生成这个插件项目的代码。

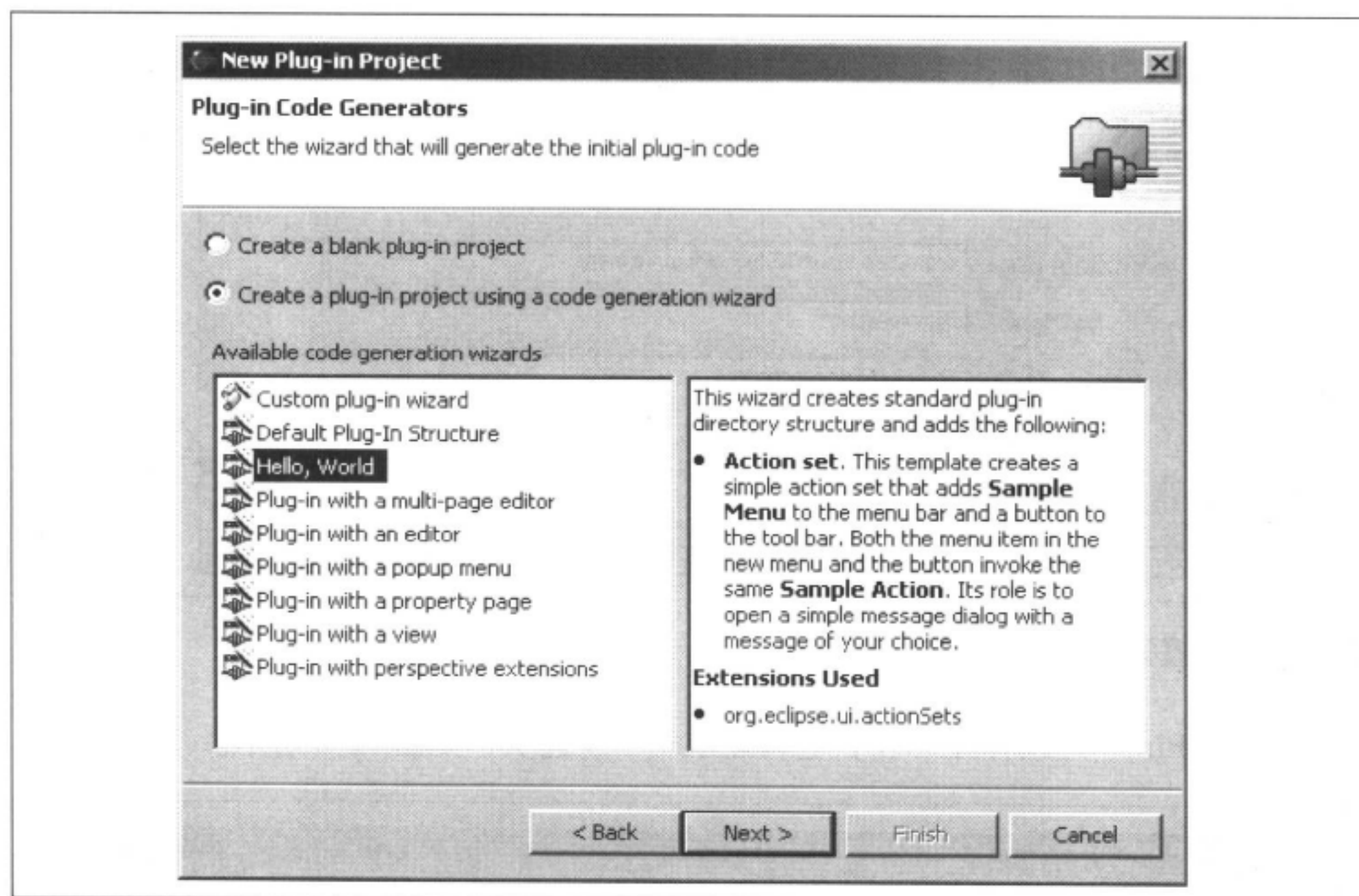


图 12-5：选择一个向导

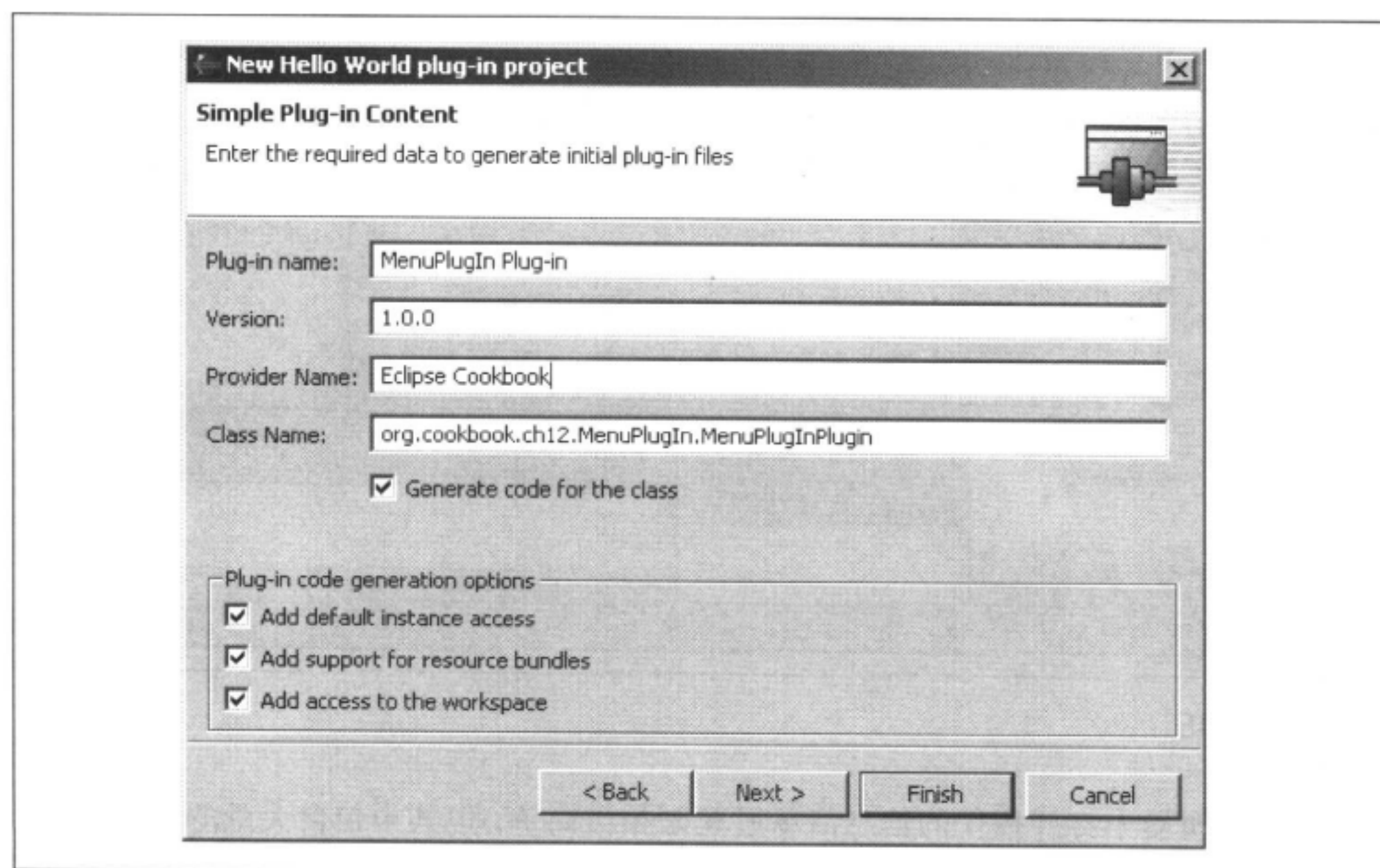


图 12-6：配置插件

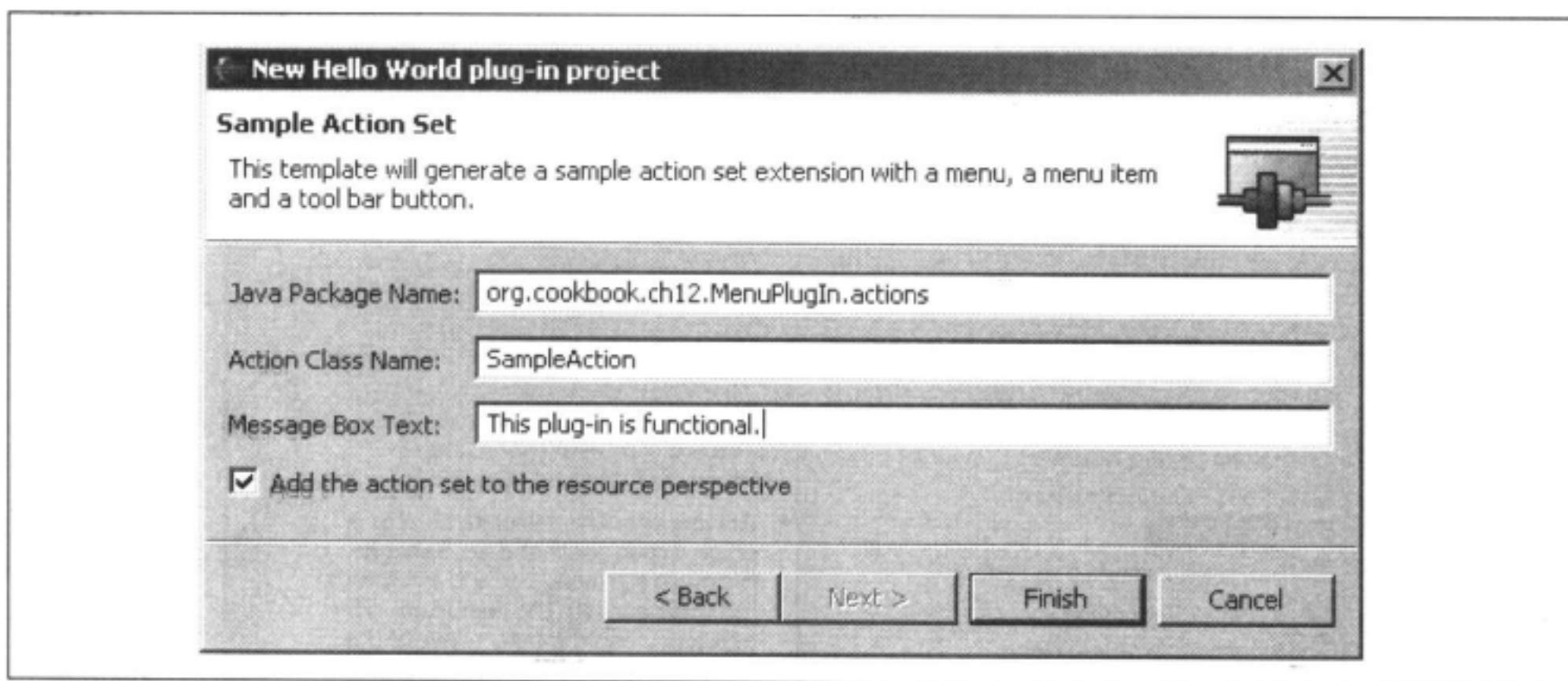


图 12-7：设置插件的显示文本

最后，单击 Finish，以打开 PDE，它将显示一个欢迎页，如图 12-8 所示。

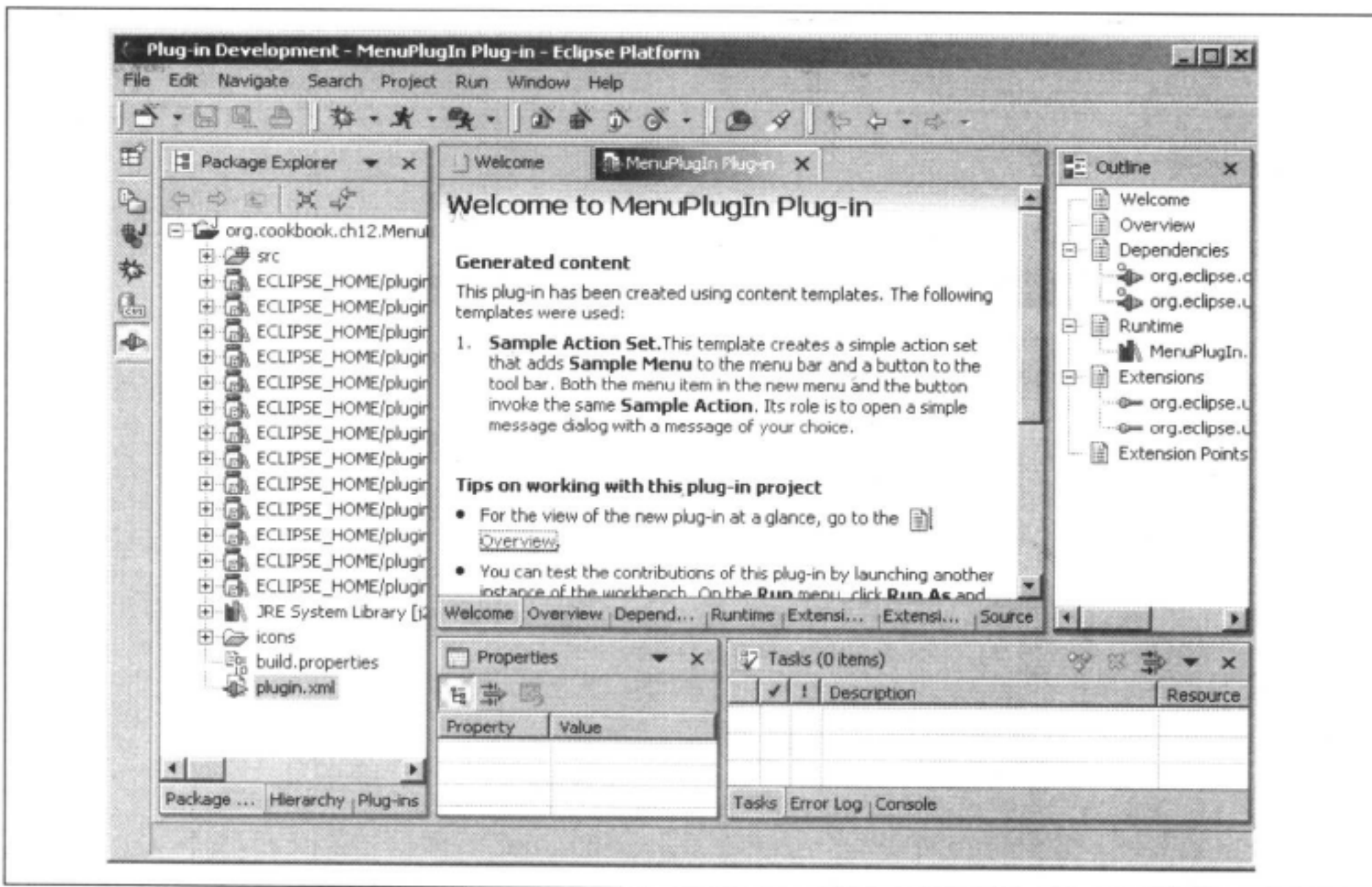


图 12-8：PDE

在这样的编辑器中打开插件的信息清单看起来相当简单，但其中包含大量的信息。这个编辑器的设计是为了便于使用信息清单，而不必转到原始的XML。注意编辑器底部的那些标签。其中包括如下标签：

Welcome

此页描述插件，如图 12-8 所示。

Overview

此页显示插件的摘要信息（包括插件名称、版本、供应商等等）。

Dependencies

此页详细说明运行该插件所必需的插件。

Runtime

此页详细说明运行该插件所需的库文件。

Extensions

此页详细说明基于其他插件的插件使用的扩展。

Extension Points

此页详细说明插件定义的扩展点。

Source

这是 *plugin.xml* 的源文件，可以直接在一个 XML 编辑器中进行编辑。

当开始在插件中添加自定义功能时，将返回到信息清单编辑器。

Eclipse 3.0

如你所料，在 Eclipse 3.0 中，PDE 得到了扩展。例如，PDE 的 Plug-in Project 创建向导增加了一个使用 OSGi 捆绑信息清单（bundle manifest）创建插件的选项（建议仅用于使用新的基于 OSGi 的 Platform Runtime 的功能的插件）。另外，PDE 有一个新的编译配置编辑器，这意味着，如果需要使用一个 *build.properties* 文件，PDE 可以提供一个专门的编译配置编辑器，加速编译过程。

参考

12.2 节，创建 *plugin.xml* 文件；12.4 节，使用运行时工作台测试插件；*Eclipse* (O'Reilly) 一书的第 11 章。

12.4 使用运行时工作台测试插件

问题

你正在开发一个插件，而且你不想在每次测试插件时都要停止并重新启动 Eclipse。

解决方案

启动一个运行时工作台，这将打开一个新的工作台，其中包含已经安装的要进行测试的插件。

讨论

如果需要，可以编译上一节创建的插件，并将其部署到 *plugins* 目录中（参阅 12.5 节）。在部署一个插件时，所需要做的全部工作就是启动一个运行时工作台，显示已经安装的插件。

要启动运行时工作台，在 Package Explorer 视图中选中插件项目，选择 Run → Run，单击 Run 对话框左边的列表框中的 Run-time Workbench 节点，并单击 New 按钮，创建新的运行配置，如图 12-9 所示。单击 Run，接受默认设置。

注意：下次需要启动 Run-time Workbench 时，只需选择 Run → Run As → Run-time Workbench 即可。

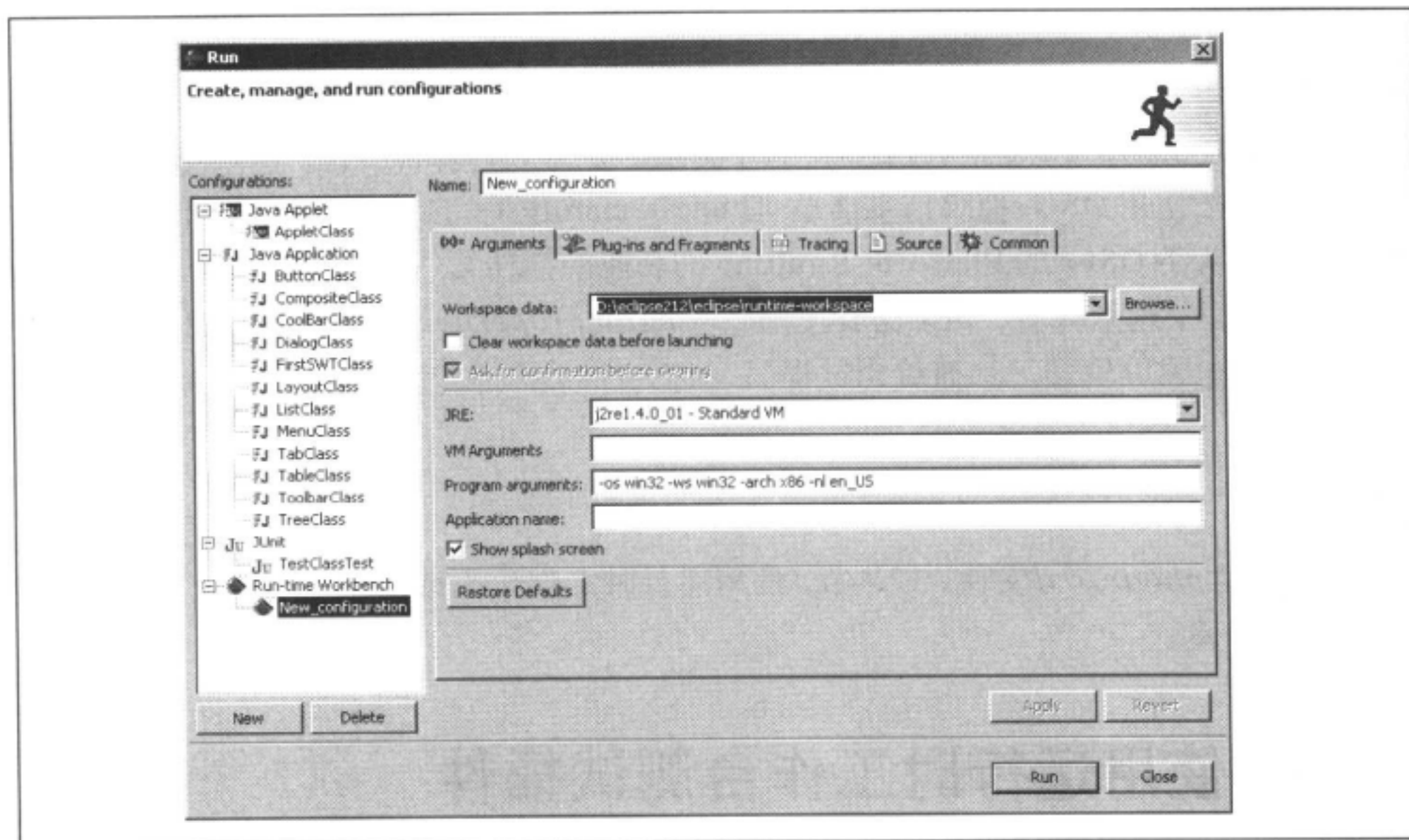


图 12-9：创建运行配置

这将启动 Run-time Workbench。从图 12-10 中可以看到这个插件定义的菜单 Sample Menu 及其添加到工具栏上的按钮（带有 Eclipse 图标）。

注意：如果在 Run-time Workbench 中看不到所开发的插件，可选择 Window → Customize Perspective → Other，选中由我们创建的插件定义的 Sample Action Set 复选框，并单击 OK。

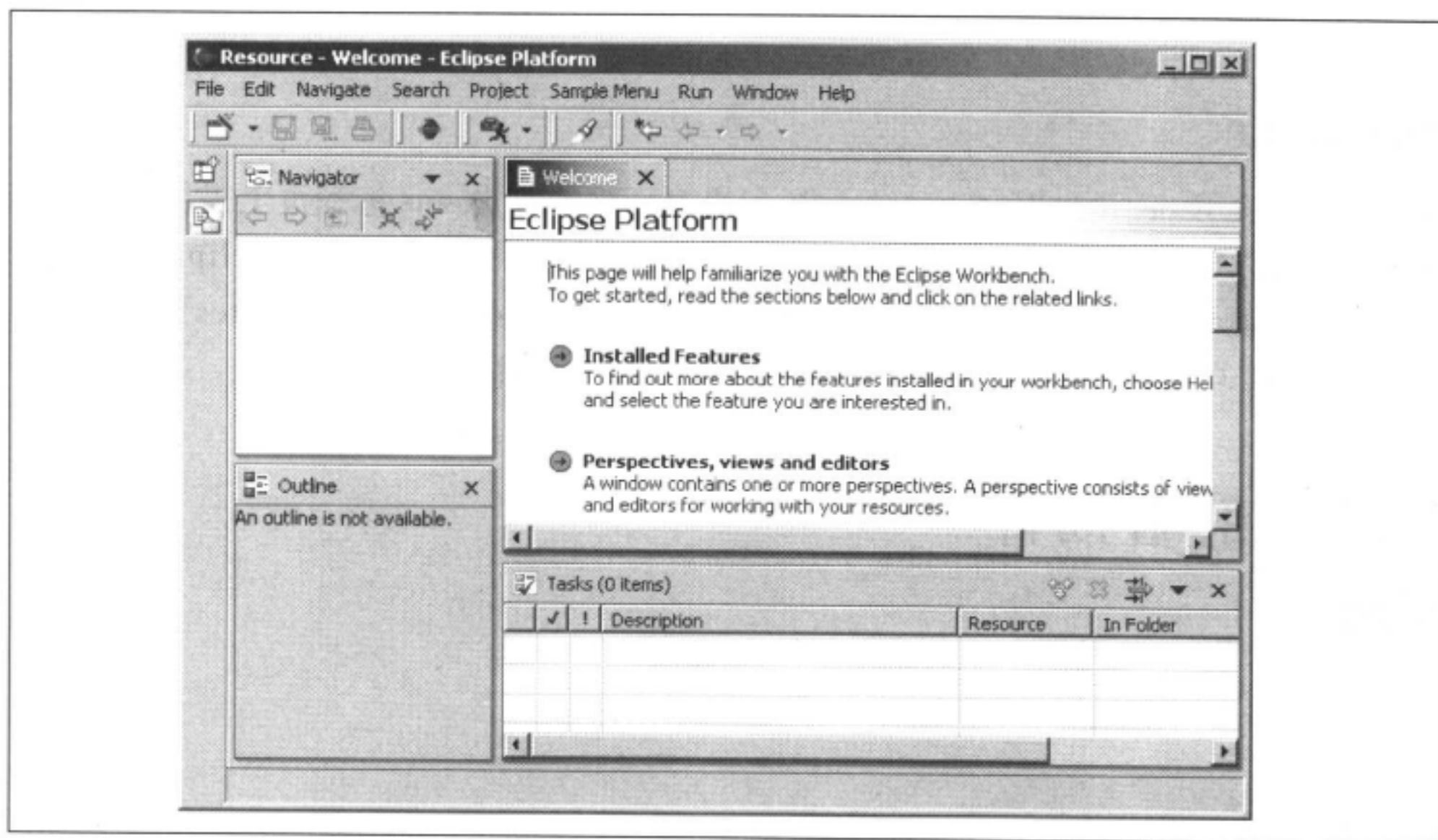


图 12-10：运行时工作台

为了证实插件可以如期运行，可选择 Sample Menu → Sample Action，或者单击 New 按钮。这个插件将显示一个消息框，其中包含消息“This plug-in is functional.”，如图 12-11 所示。

好极了！你现在已经能够创建并运行一个 Eclipse 插件了。要退出插件，可关闭 Run-time Workbench。

注意：正如你所期待的，可以调试插件代码；只需选择 Run → Debug As → Run-time Workbench 即可。

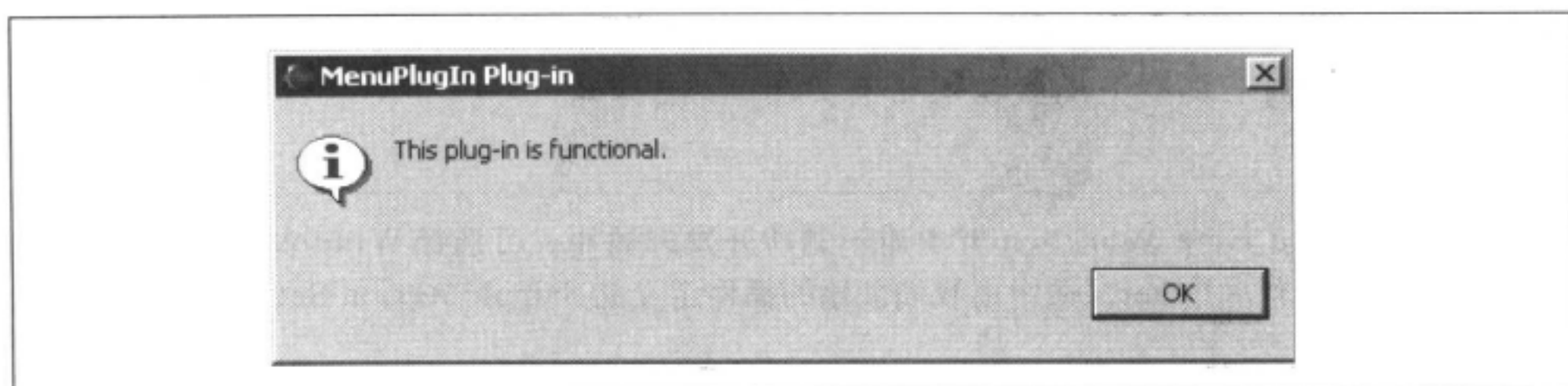


图 12-11：插件的消息

Eclipse 3.0

Eclipse 3.0 在 Run-time Workbench 运行配置对话框的 Plug-ins 标签页上提供了一个新的选项，使你能够在测试插件时选择要包含的一组插件。就测试而言，Eclipse 3.0 PDE 有一个新的测试平台，用于插件的基于 JUnit 的单元测试组（unit test suites）。事实上，这是 Eclipse 3.0 的一个重大改进：可以使用 JUnit 测试插件。

12.5 部署插件

问题

你想部署所开发的插件。

解决方案

右击插件项目，单击 Export，然后按照说明进行操作。Eclipse 会处理细节问题。

讨论

要部署插件，右击插件项目，并单击 Export。在 Export 对话框中选中 Deployable plug-ins and fragments 复选框，并单击 Next，以打开 Export Plug-ins and Fragments 对话框，如图 12-12 所示。选择 `org.cookbook.ch12.MenuPlugIn (1.0.0)`。要创建一个单独的可部署的 .zip 文件，使用 Browse 按钮浏览到 `eclipse/workspace/org.cookbook.ch12/MenuPlugIn`；Eclipse 将添加文件名 `MenuPlugIn_1_0_0.zip`。然后单击 Finish。

单击 Finish，创建准备用于部署的 `MenuPlugIn_1_0_0.zip` 文件。这个文件中包含两个文件：`MenuPlugIn.jar` 和 `plugin.xml`。这个 .zip 文件，即 `MenuPlugIn_1_0_0.zip` 文件，就是要向用户部署的文件——用户所需要做的就是解压这个文件。

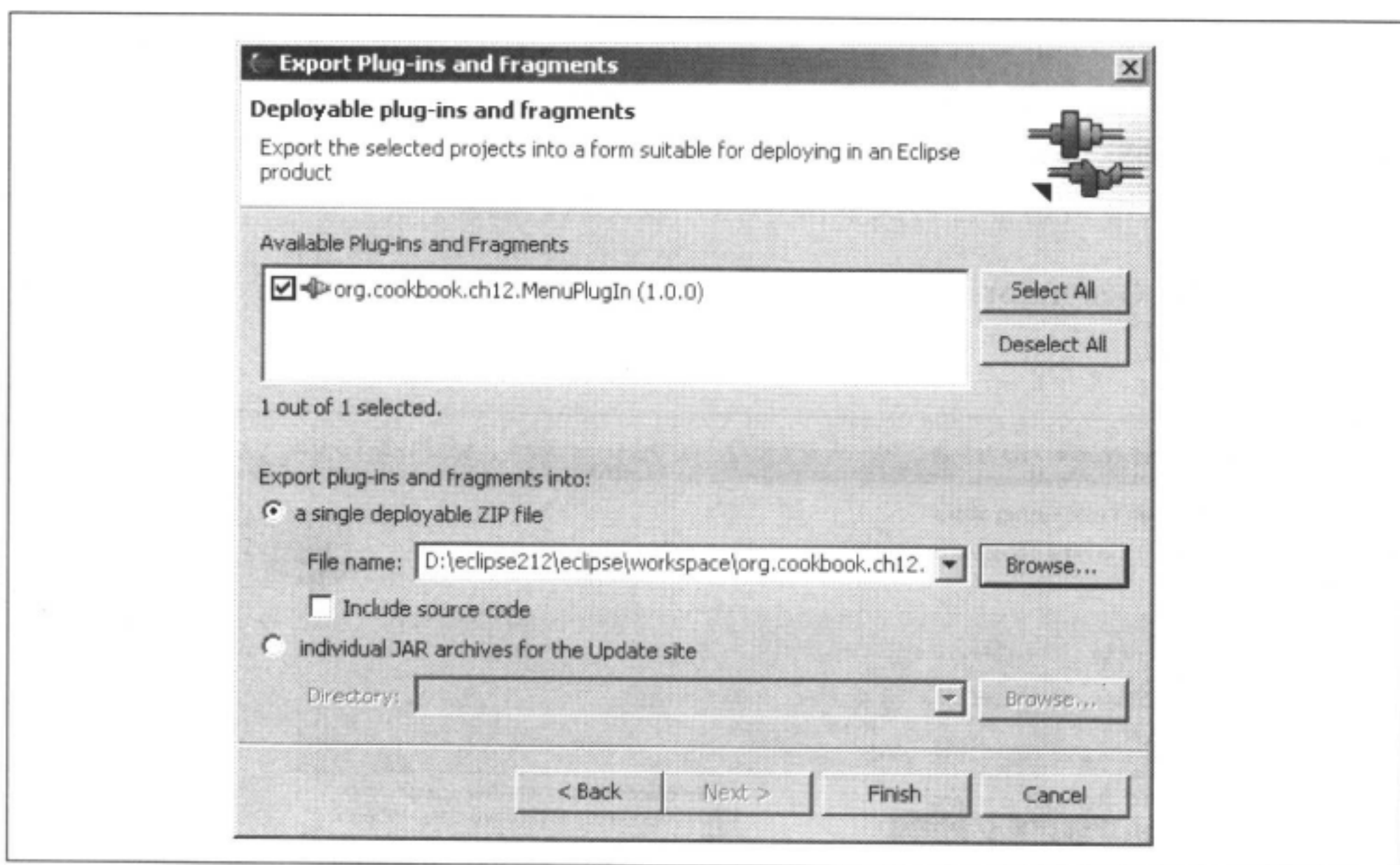


图 12-12：导出插件

注意：在 New Plug-in Project 对话框的第三页中首次创建项目时，可以指定项目要创建的 *.jar* 文件的名称。

12.6 使用框架编写插件

问题

你想自己从头开发插件的代码。

解决方案

可以通过在 Plug-in Code Generators 页选择 Default Plug-In Structure，创建一个框架插件项目。本节将创建一个框架插件，并以此为起点开发插件的其余部分。

讨论

作为一个例子，我们将创建本章早些时候创建的显示菜单的插件，但这一次，我们将自己动手，而不是完全依赖于向导。然而，我们仍然要从向导开始，所以选择 File → New

→ Project。在 New Project 对话框中，在左边的列表框中选择 Plug-in Development，而在右边的列表框中选择 Plug-in Project，并单击 Next。在下一个对话框中，输入新项目的名称“org.cookbook.ch12.MenuPlugInFromScratch”，并单击 Next，以打开 Plug-in Project Structure 对话框。再次单击 Next，接受默认设置。

在 Plug-in Code Generators 对话框中，选择 Default Plug-In Structure，如图 12-13 所示，并单击 Next。

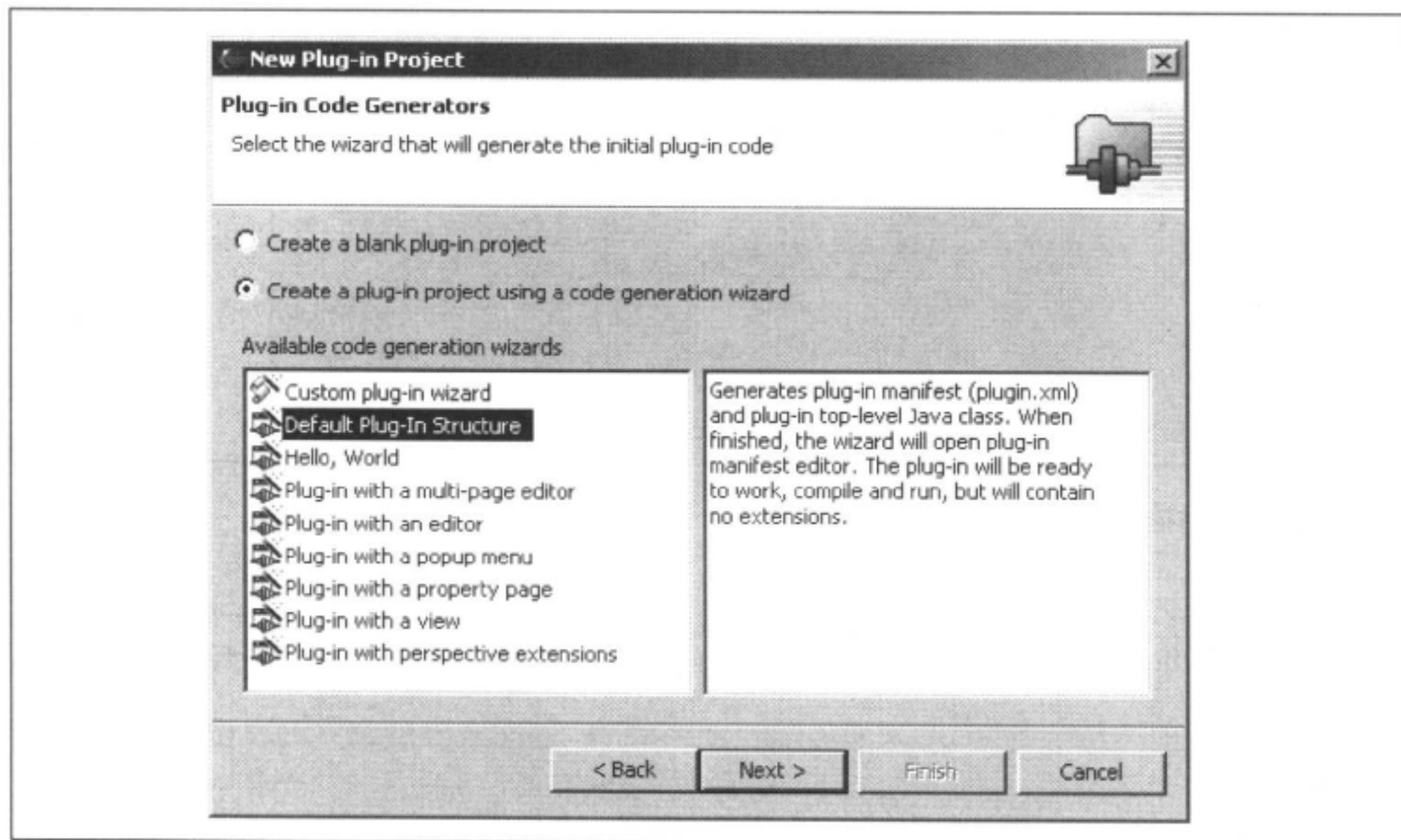


图 12-13：选择插件向导

在下一个对话框中，可设置供应商的名称，在此设置为 Eclipse Cookbook。由于我们打算自己编写代码，所以不要选择 Plug-in code generation options 选项组中的复选框，如图 12-14 所示，然后单击 Finish。

单击 Finish，创建 MenuPlugInFromScratch 插件，同时在 Eclipse 中打开其信息清单。单击 Source 标签，显示 *plugin.xml* 文件中的 XML 代码。该 XML 代码应该如下所示：

```
<?xml version="1.0" encoding="UTF-8"?>
<plugin
  id="org.cookbook.ch12.MenuPlugInFromScratch"
  name="MenuPlugInFromScratch Plug-in"
  version="1.0.0"
  provider-name="Eclipse Cookbook"
  class="org.cookbook.ch12.MenuPlugInFromScratch.MenuPlugInFromScratchPlugin">
```

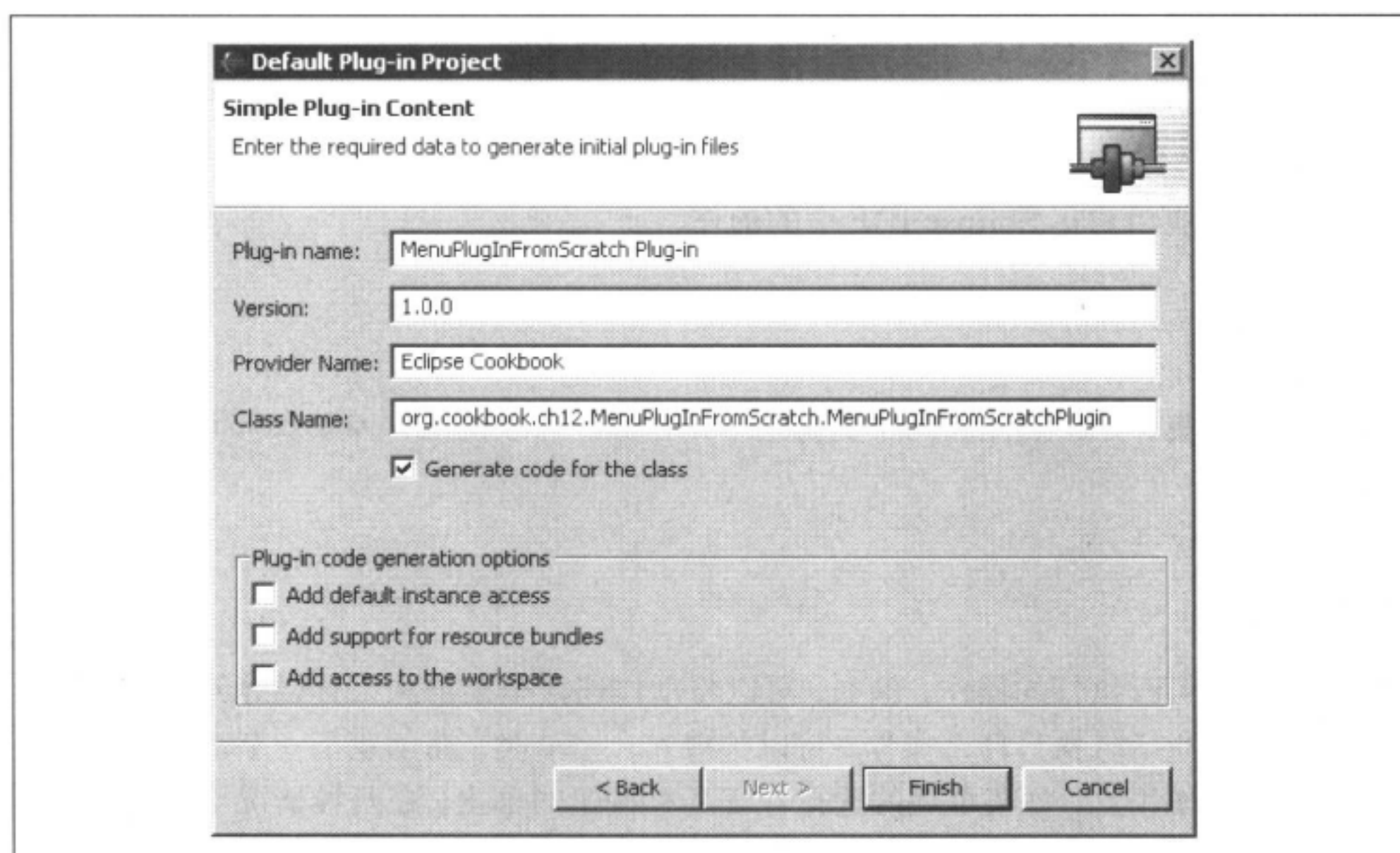



图 12-14：设置供应商的名称

```
<runtime>
  <library name="MenuPlugInFromScratch.jar"/>
</runtime>
<requires>
  <import plugin="org.eclipse.core.resources"/>
  <import plugin="org.eclipse.ui"/>
</requires>

</plugin>
```

这就是我们新建的插件的信息清单。插件的目标是增加一个新的菜单项，并在工具栏上增加一个按钮。为了在插件中实现上述目标，需要定义动作和动作集。详细内容请参阅下一节。

参考

12.2 节，创建 *plugin.xml* 文件；12.9 节，创建动作集；12.10 节，编写 Eclipse 动作的代码；*Eclipse* (O'Reilly) 一书的第 11 章和第 12 章。

12.7 在插件中响应用户的动作

问题

你想让插件响应用户在 Eclipse IDE 中的动作。

解决方案

创建一个动作集，并在插件代码中支持这个动作集。可以使用 New Extension 向导来完成这一任务。

讨论

动作表示用户可以执行的操作；例如，可以将动作连接至菜单和工具栏等元素。要指定当一个动作被激活时执行什么操作，可以扩展 `Action` 类。在创建了一个动作之后，可以在许多场合使用它，如菜单项的选择和工具栏按钮的单击；在两种情况下，同一个动作对象将执行同样的操作。顾名思义，动作集就是一组动作。使用动作集，可以将动作连接到菜单项和工具栏按钮。

使用扩展点可以开发动作集。扩展点可以使一个插件基于另一个插件的导出。能够实现菜单和工具栏动作的动作集是 `org.eclipse.ui.actionSets` 扩展点的扩展。

注意： 插件只能使用由其他插件导出的类，这使得扩展点变得特别重要。Eclipse 自带的几个标准插件中内置了对自定义插件的大量支持。为了使一个插件使用你的 Java 代码，可以把该代码放到一个 `.jar` 文件中；然后将这个 `.jar` 文件包含在一个插件中，以便其他插件能够访问这些代码。

要创建将一个菜单和工具栏按钮连接到一个动作的动作集，可在 *MenuPlugInFromScratch* 项目的插件信息清单编辑器中单击 `Extension` 标签，并单击 `Add`，打开 New Extension 向导，如图 12-15 所示。在左边的列表框中选择 `Generic Wizards`，而在右边的列表框中选择 `Schema-based Extension`，然后单击 `Next`。

在下一个对话框中，选择扩展所需要基于的扩展点，如图 12-16 所示。由于要创建一个支持工具栏按钮和菜单项的动作集，所以使用 `org.eclipse.ui.actionSets` 扩展点。选中该扩展点，如图 12-16 所示，然后单击 `Finish`。

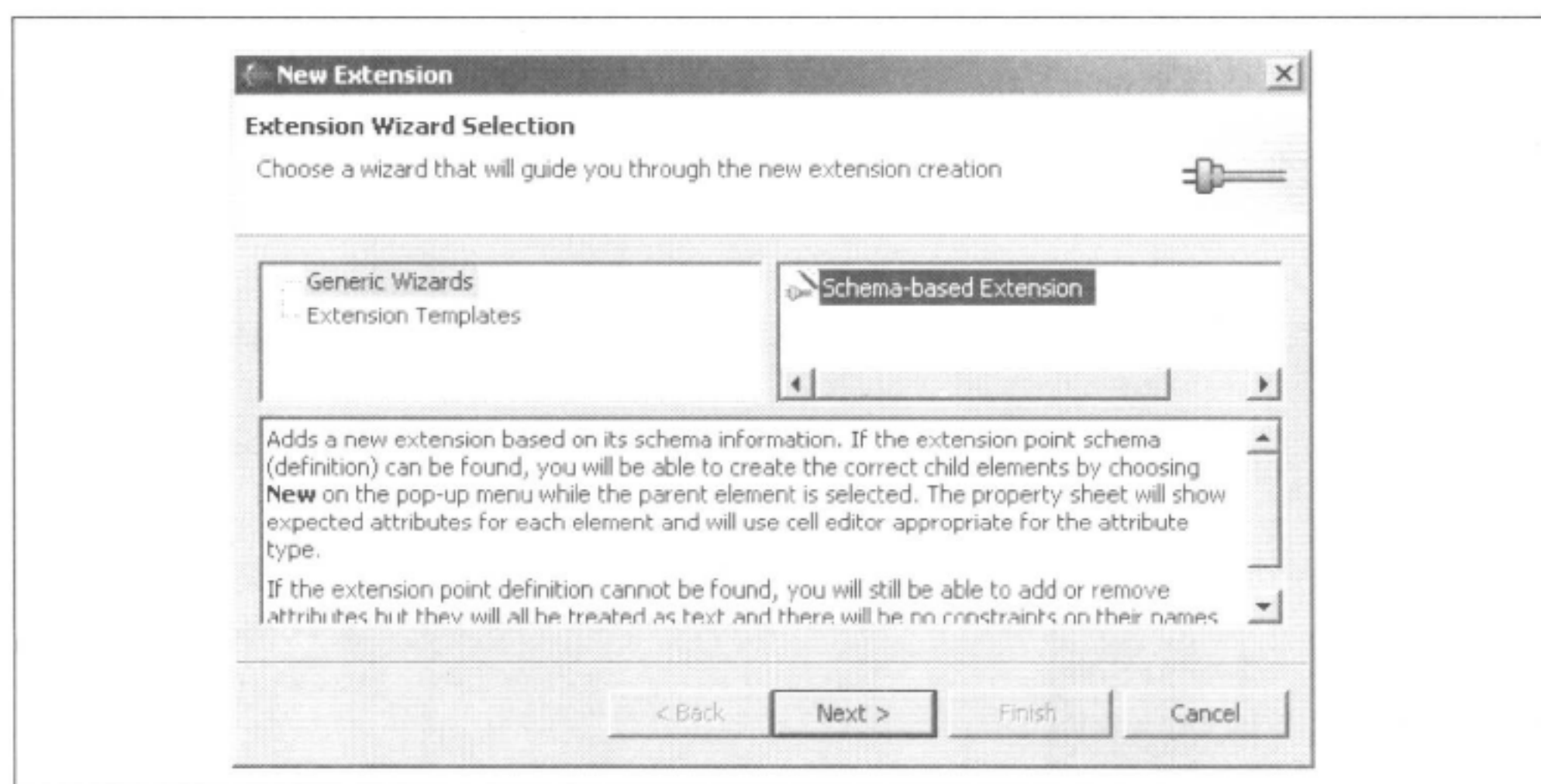


图 12-15：使用 New Extension 向导

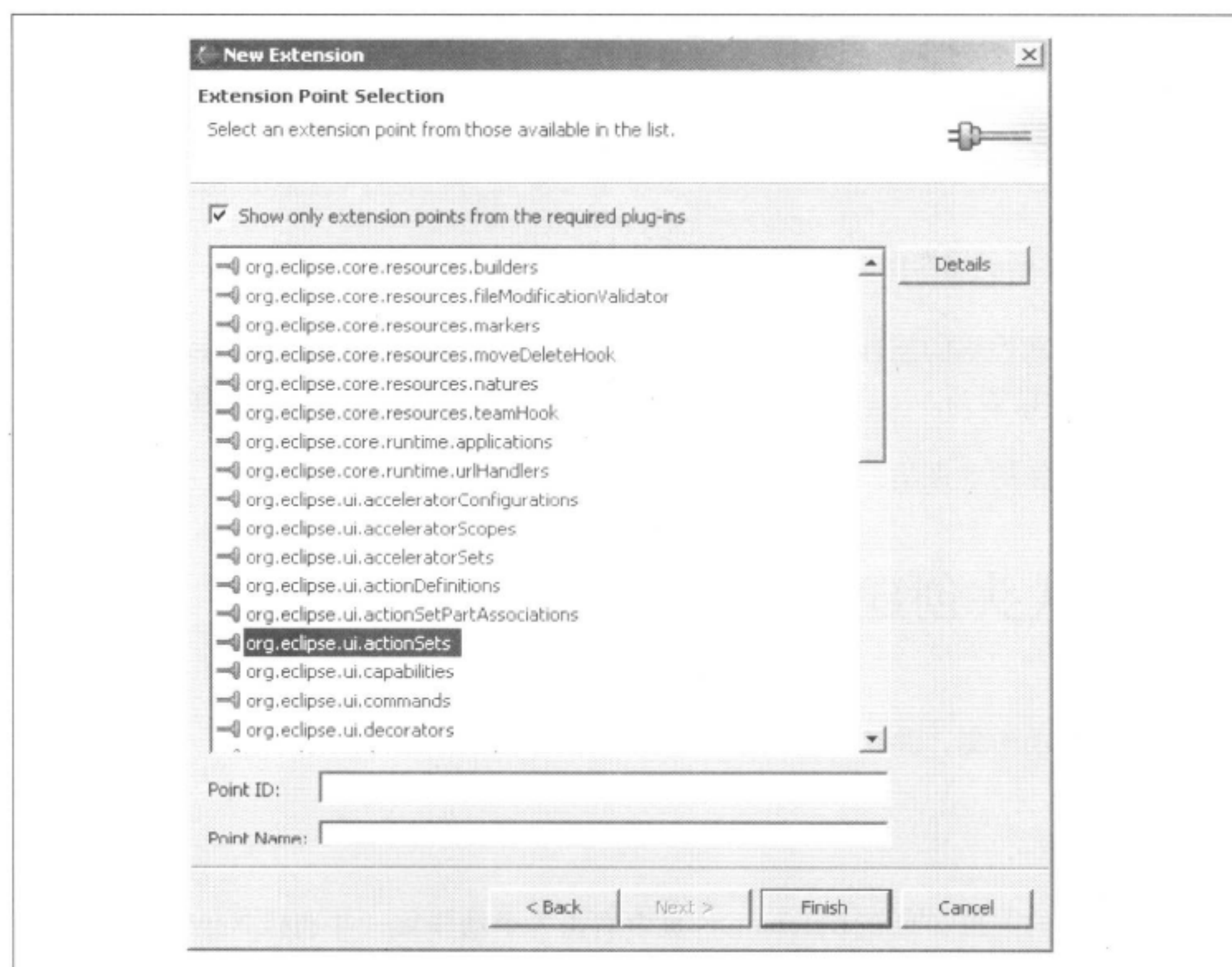


图 12-16：选择一个扩展点

当单击 Finish 时，一个代表 `org.eclipse.ui.actionSets` 扩展点的条目被添加到 Eclipse 信息清单编辑器的 Extensions 页中，如图 12-17 所示。要创建新的动作集，可在 All Extensions 列表框中右击 `org.eclipse.ui.actionSets`，并选择 New → actionSet，创建一个新的动作集，如图 12-17 所示。

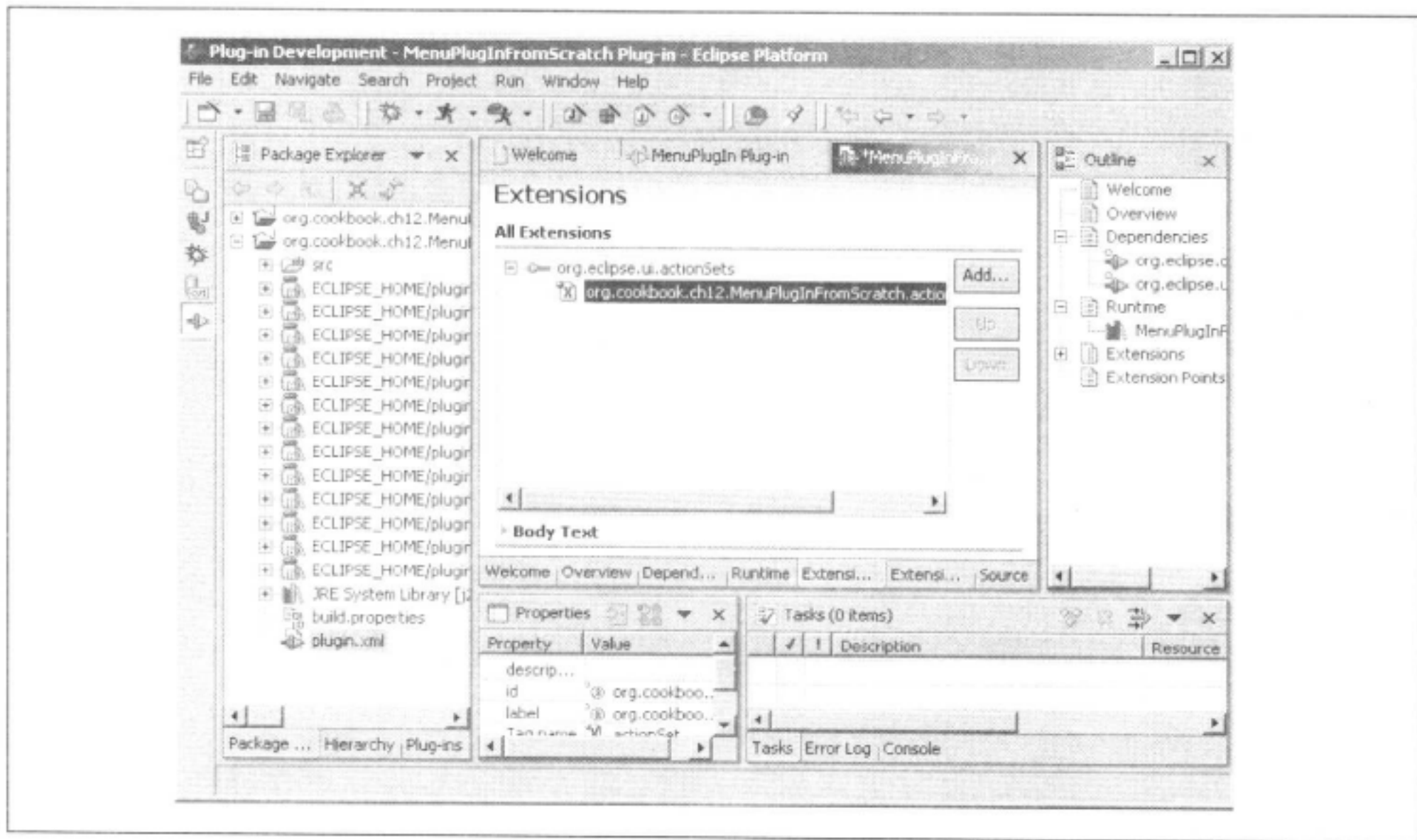


图 12-17：一个新的动作集

要设置这个新动作集的属性，可选中它，然后在 Eclipse 窗口底部的 Properties 视图中，将 label 属性设置为 Action Set 1，将 visible 属性设置为 true，如图 12-18 所示。

这样就创建了新的动作集。

12.8 从头创建插件菜单

问题

你有一个插件动作集，并且想使用它创建一个新的菜单。

解决方案

要创建一个新的菜单，可在 Extensions 页中右击一个动作集，并选择 New → Menu。

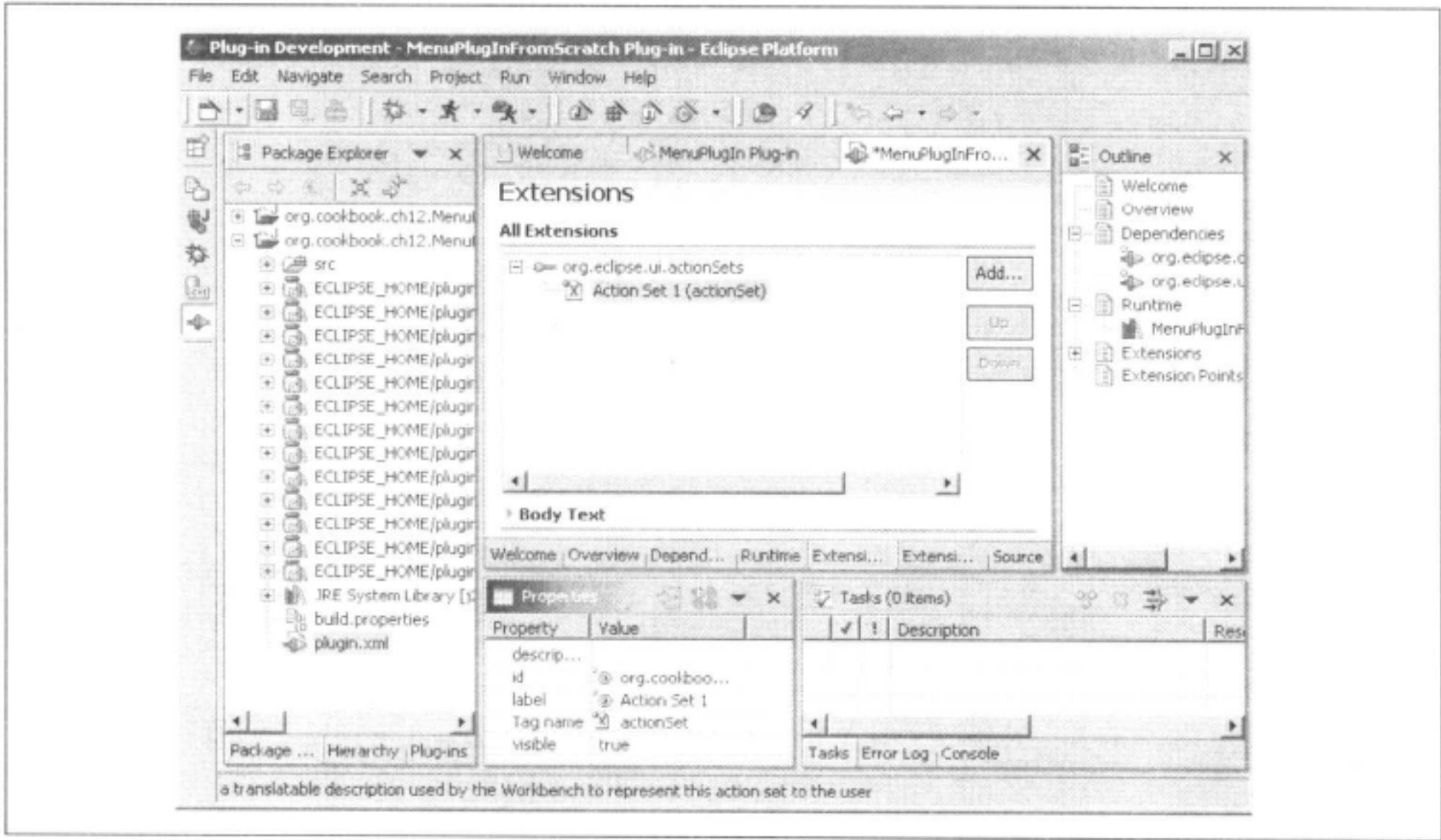


图 12-18：配置动作集

讨论

我们将继续上一节中开发的示例，并在上一节创建的动作集中添加一个新的菜单。在 Extensions 页右击 Action Set 1，并选择 New → Menu。在 Properties 视图中，把新菜单的 id 属性设置为 Menu1，把 label 属性设置为 Menu 1，如图 12-19 所示，在 Action Set 1 下方出现了这个新建的菜单。

我们打算在这个菜单中添加一个菜单分隔符，作为占位符，以便以后在该菜单中添加其他菜单项，并把菜单项组和在一起。要创建一个菜单分隔符，可右击菜单 Menu 1，并选择 New → Separator。在 Properties 视图中，把分隔符的 name 属性设置为 Group1，如图 12-20 所示。

这将创建一个新菜单，并把它添加到动作集中。现在，你需要创建一些菜单项，并把一个动作连接到这些菜单项，以便执行某些操作。详细的介绍请参阅下一节。

参考

12.9 节，创建动作；12.10 节，为插件动作编写代码；*Eclipse* (O'Reilly) 一书的第 11 章和第 12 章。

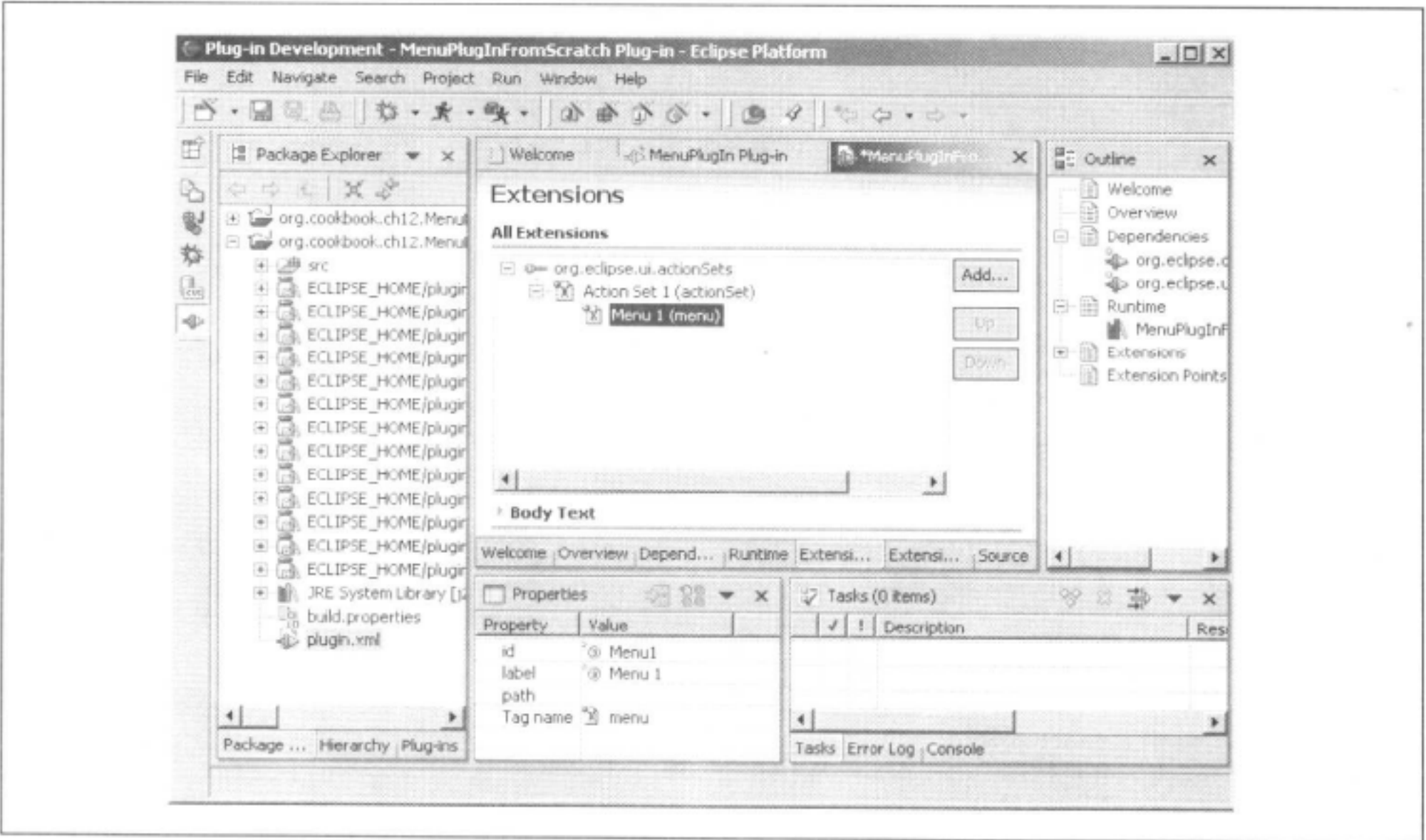


图 12-19：一个新的菜单

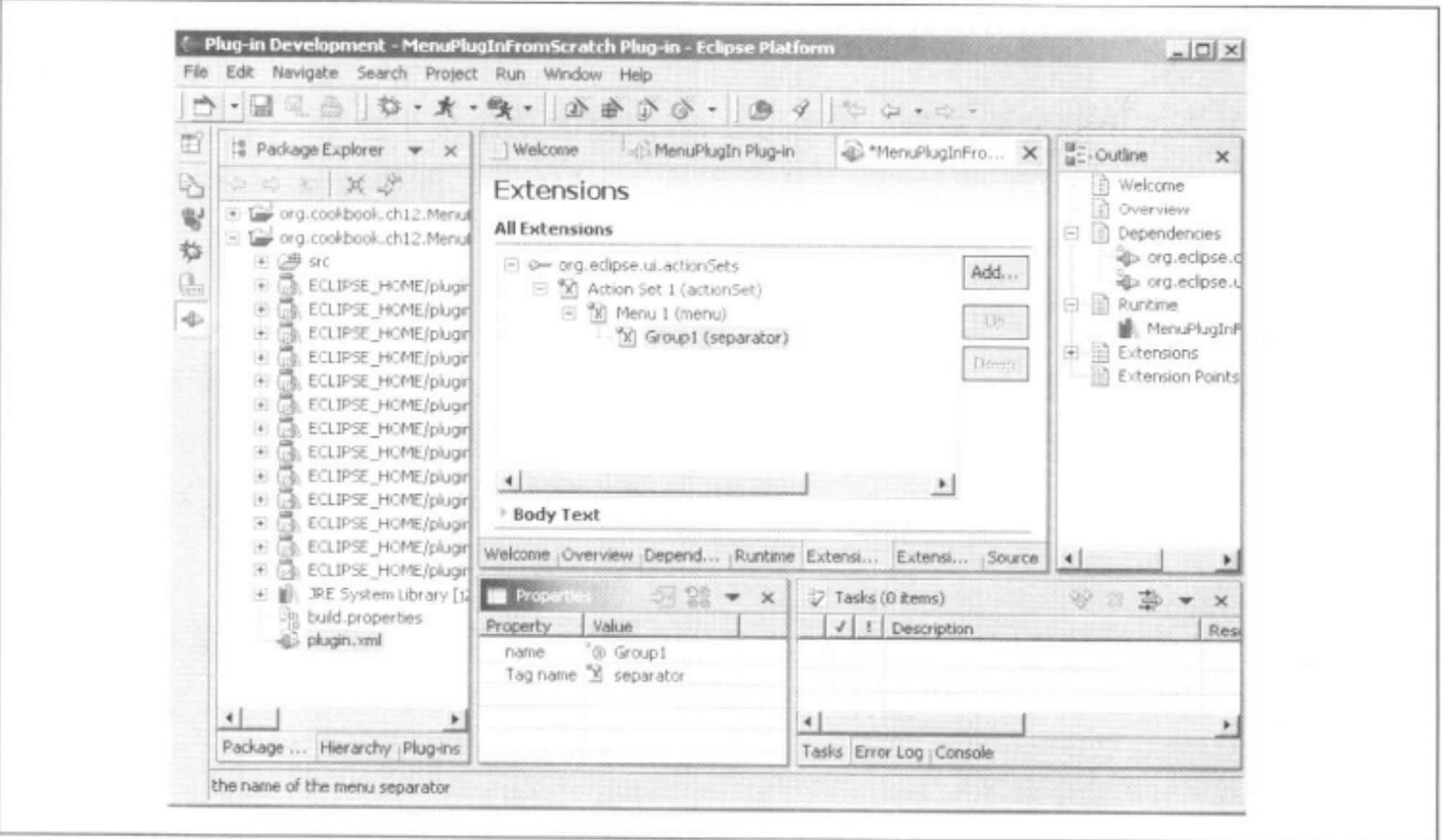


图 12-20：一个新的菜单分隔符

12.9 创建动作

问题

你想创建一些菜单项和工具栏按钮，并把 Java 代码连接到这些菜单项和工具栏按钮。

解决方案

新建一个动作，添加菜单项和工具栏按钮，并用 Java 代码实现所需的功能。

讨论

在前两节开发的示例的基础上，在插件中增加一些菜单项和一个工具栏按钮。在插件信息清单编辑器的 Extensions 页上，右击 Action Set 1，并选择 New → Action。在 Properties 视图中，把 label 属性设置为 Action 1，把 tooltip 属性设置为 “This action is functional.”，而 id 属性设置为 Action1。接下来，把 menubarPath 属性设置为 Menu1/Group1，而 toolbarPath 属性设置为 Group1。这样就可以把该动作连接到菜单项和工具栏按钮。

为了把这个动作连接到代码，在 Extensions 页上选择动作 Action 1，并单击省略号 (...) 按钮——当在 Properties 视图中选中 class 属性时将出现省略号按钮。这将打开 Java Attribute Editor 对话框，如图 12-21 所示。

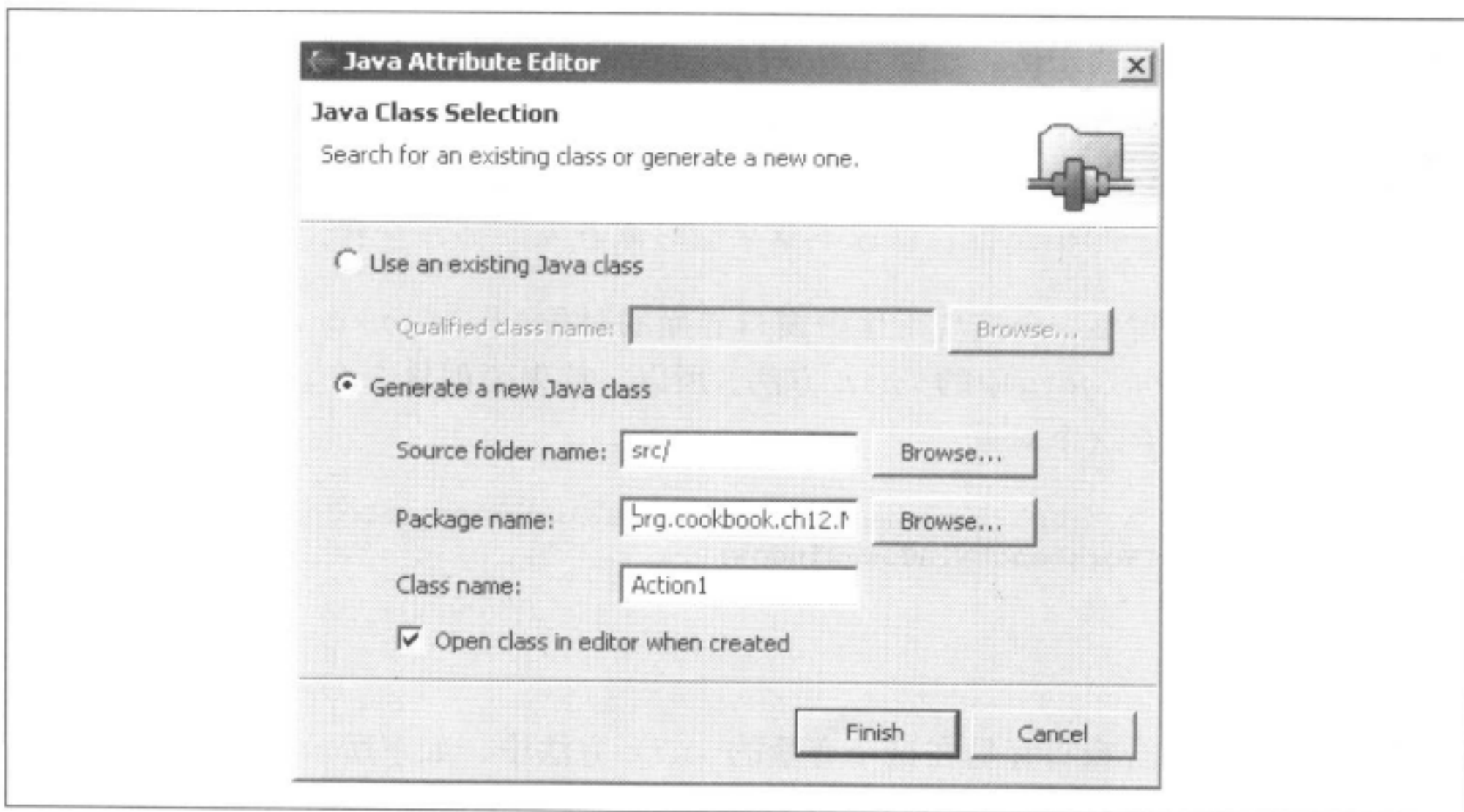


图 12-21：创建一个动作类

选中 **Generate a new Java class** 单选按钮，在 **Package name** 文本框中输入 `org.cookbook.ch12.MenuPlugInFromScratch`（或者单击 **Browse** 按钮，并选择这个包），把这个类命名为 `Action1`，如图 12-21 所示。然后单击 **Finish**。这将创建这个新动作，并把它的 Java 支持文件 `Action1.java` 添加到项目中。

参考

12.7 节，在插件中响应用户的动作；12.10 节，为插件动作编写代码；*Eclipse* (O'Reilly) 一书的第 11 章。

12.10 为插件动作编写代码

问题

你有一个动作，而且你需要让它实际发挥作用。

解决方案

编辑动作的 `.java` 文件，并实现所需的支持。

讨论

继续前面 3 节中开发的示例，如果 `Action1.java` 尚未打开的话，那就打开它。这个文件中包含大量的 `TODO` 语句，比如，将代码添加到动作的构造函数中以自定义动作；在本例中，我们将忽略这些 `TODO` 语句。在这个例子中，当用户选择一个菜单项或单击工具栏按钮时，将显示一个消息框，就像本章早些时候开发的插件一样。

为了显示消息框，需要一个实现工作台窗口界面的对象，即 `IWorkbenchWindow`。这个对象被传递给 `Action1.java` 中的 `init` 方法，所以，首先要创建一个 `private` 类型的变量 `window`，以保存这个对象：

```
public class Action1 implements IWorkbenchWindowActionDelegate {  
    private IWorkbenchWindow window;  
    .  
    .  
    .  
}
```

然后把显示的工作台窗口存储在这个变量的 `init` 方法中，如下所示：

```
public void init(IWorkbenchWindow window) {  
    this.window = window;  
}
```

可以在动作的run方法中,使用这个变量和MessageDialog类的openInformation方法,显示一个包含消息“This plug-in is functional.”的消息框。要使用MessageDialog类,首先应导入它,然后在run方法中调用它:

```
import org.eclipse.jface.dialogs.MessageDialog;  
.  
.  
.  
public void run(IAction action) {  
    MessageDialog.openInformation(  
        window.getShell(),  
        "New Plug-in",  
        "This plug-in is functional.");  
}
```

这样就完成了这个插件。现在,保存所有文件,并重新编译项目。要测试这个新的插件,应启动Run-time Workbench。

注意,要加载这个插件,还需要一个步骤,而对于本章早些时候使用向导开发的插件来说,这一步骤不是必需的。必须把这个插件明确地添加到当前透视图,以便能够看到这个插件,所以,应在Run-time Workbench中选择Window → Customize Perspective,选中Action Set 1复选框,如图12-22所示,并单击OK。

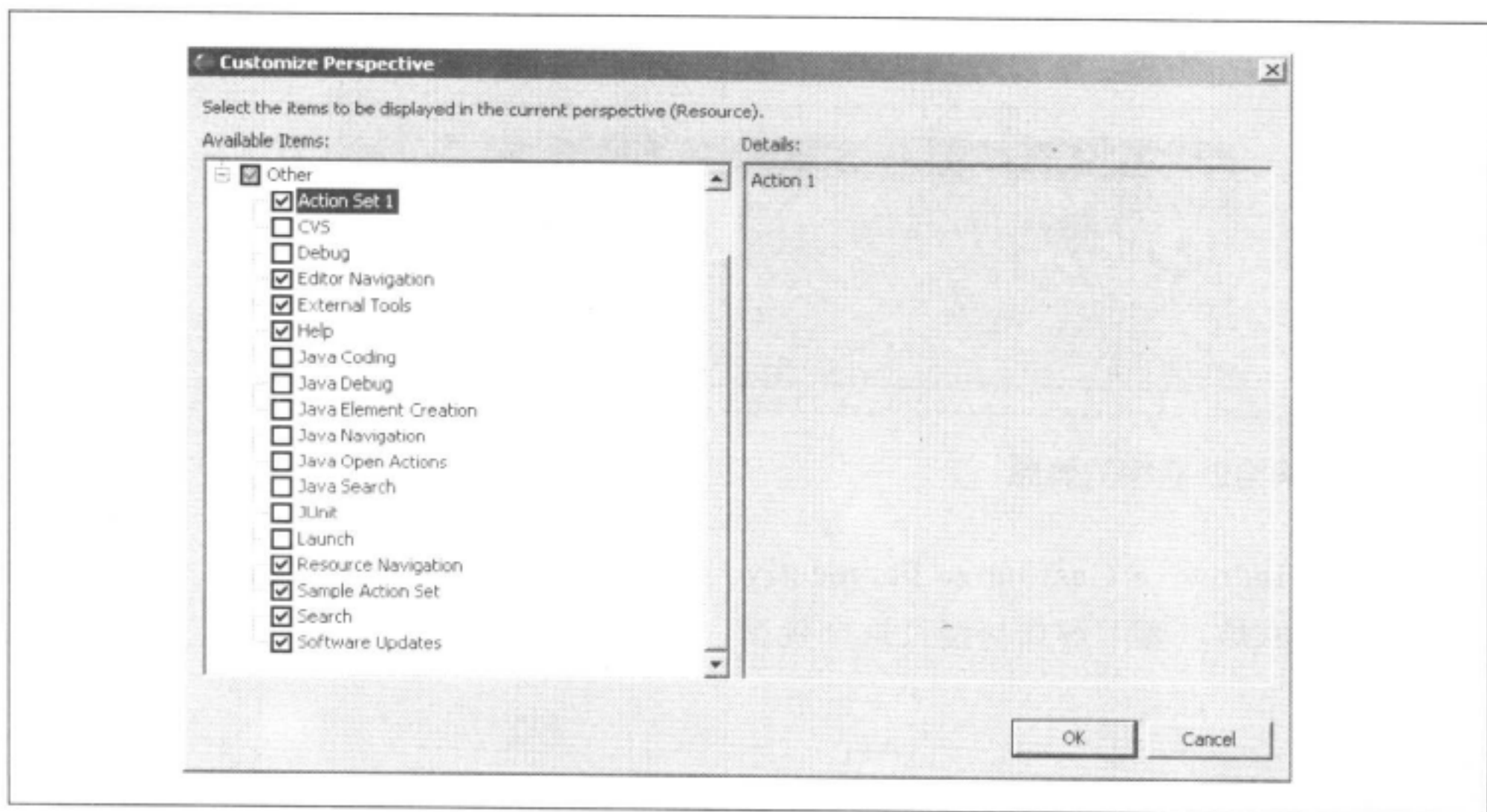


图 12-22: 自定义透视图

现在,应该可以看到Eclipse用于工具栏按钮的默认方形按钮图标以及新的菜单Menu 1,如图 12-23 所示。

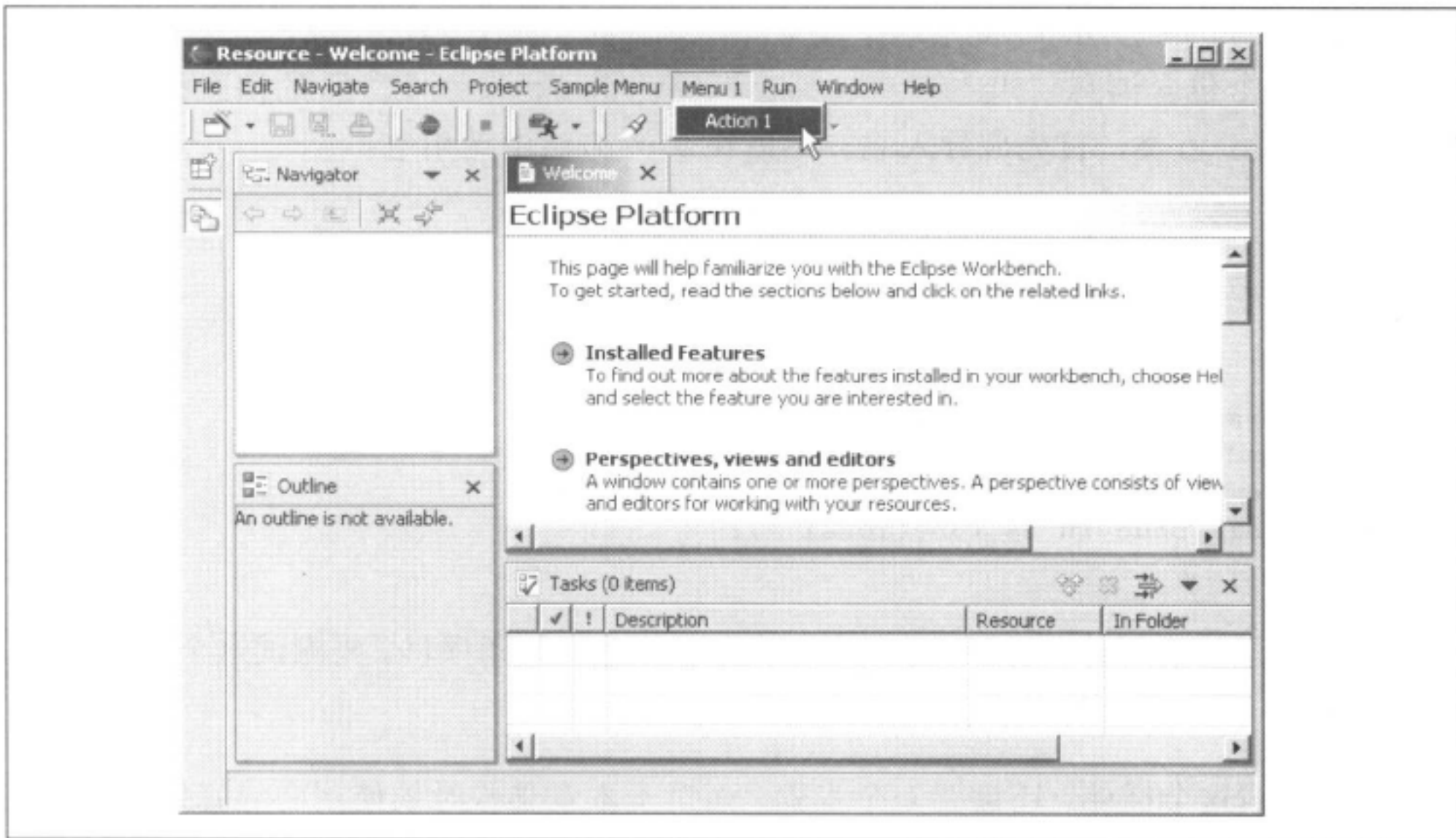


图 12-23: 一个新的有效的插件

选择新的菜单 Menu 1 → Action 1, 或单击 New 按钮时, 应该可以看到 New Plug-in 消息框, 如图 12-24 所示。恭喜! 你是一名 Eclipse 插件开发人员了。

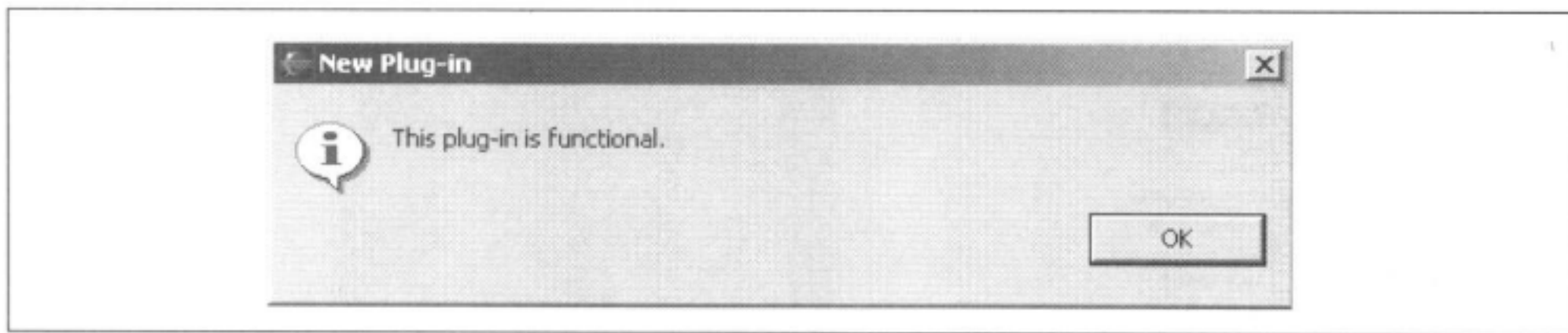


图 12-24: 来自新插件的消息

每次使用 Window → Customize Perspective 菜单启动一个新插件时必须自定义这个视图, 这有些麻烦。要了解如何绕过这个视图, 请参阅下一节。

参考

12.7 节, 在插件中响应用户的动作; 12.9 节, 创建动作; 12.11 节, 将插件自动添加到透视图。

12.11 将插件自动添加到透视图问题

你不想让用户必须通过自定义方式明确地将插件添加到透视图。

解决方案

在`plugin.xml`文件中使用扩展点`org.eclipse.ui.perspectiveExtensions`将插件自动添加到透视图。

讨论

要把插件自动添加到透视图，可在`plugin.xml`文件中使用扩展点`org.eclipse.ui.perspectiveExtensions`。只需编辑`plugin.xml`文件中的XML代码即可。下面的代码说明了如何在`plugin.xml`文件将前几节开发的插件自动添加到Java透视图：

```
<?xml version="1.0" encoding="UTF-8"?>
<plugin
    id="org.cookbook.ch12.MenuPlugInFromScratch"
    name="MenuPlugInFromScratch Plug-in"
    version="1.0.0"
    provider-name="Eclipse Cookbook"
    class="org.cookbook.ch12.MenuPlugInFromScratch.MenuPlugInFromScratchPlugin">

    <runtime>
        <library name="MenuPlugInFromScratch.jar"/>
    </runtime>
    <requires>
        <import plugin="org.eclipse.core.resources"/>
        <import plugin="org.eclipse.ui"/>
    </requires>

    <extension
        point="org.eclipse.ui.actionSets">
        <actionSet
            label="Action Set 1"
            visible="true"
            id="org.cookbook.ch12.MenuPlugInFromScratch.actionSet1">
            <menu
                label="Menu 1"
                id="Menu1">
                <separator
                    name="Group1">
                </separator>
            </menu>
            <action
```

```
        label="Action 1"
        class="org.cookbook.ch12.MenuPlugInFromScratch.Action1"
        tooltip="This action is functional."
        menubarPath="Menu1/Group1"
        toolbarPath="Group1"
        id="Action1">
    </action>
</actionSet>
</extension>
<extension
    point = "org.eclipse.ui.perspectiveExtensions">
    <perspectiveExtension
        targetID="org.eclipse.ui.javaPerspective">
        <actionSet
            id="org.cookbook.ch12.MenuPlugInFromScratch.actionSet1">
        </actionSet>
    </perspectiveExtension>
</extension>
</plugin>
```

编辑完成之后保存 *plugin.xml* 文件，并重新启动 Run-time Workbench。这样，新的插件在启动时应出现在透视图图中。

注意： 使用类似的命名法，可以将插件添加到其他透视图图中。例如，使用扩展点 `org.eclipse.ui.resourcePerspective`，可以把插件添加到 Resource 透视图图中。

创建插件：向导、 编辑器和视图

13.0 简介

在本章中，我们将通过创建支持向导、编辑器和视图的插件，扩展插件的功能。这并没有你想象的那么难，这主要是因为我们可以依靠模板，使用 PDE 可以创建这些模板，并在必要时修改模板。本章将介绍如何使用模板来创建插件。

13.1 创建支持向导和编辑器的插件

问题

你想创建一个可以使用向导创建文件，并使用编辑器编辑文件的插件。

解决方案

使用 PDE 向导模板。新建一个插件项目，选择 Plug-in with a multi-page editor 模板，然后按照屏幕上的指示创建一个框架模板，准备进行自定义。

讨论

在这个例子中，我们要创建一个插件，Eclipse 将这个插件与文件扩展名 .new 相关联。插件的向导使用户能够创建具有这个扩展名的文件，并且当用户双击这些文件时，这些文件将出现在插件的编辑器中。

在这里，我们可以借助于 PDE 来创建插件的模板。选择 New → Project。在 New Project 对话框中，在左边的列表框中选择 Plug-in Development，而在右边的列表框中选择 Plug-

in Project, 然后单击 Next。在下一个对话框中, 将项目命名为 `org.cookbook.ch13.EditorPlugIn`, 如图 13-1 所示。

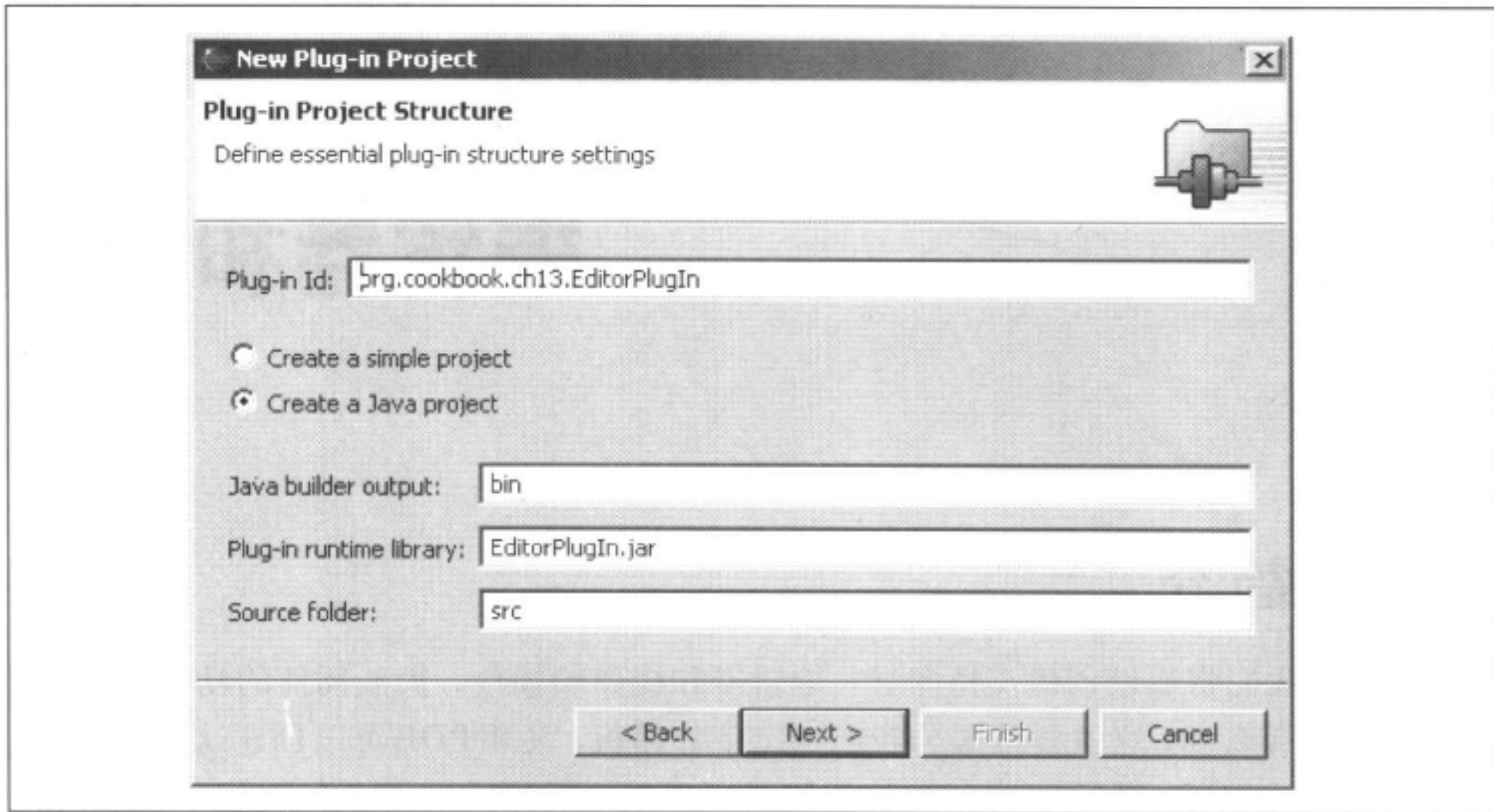


图 13-1: 为插件命名

再次单击 Next, 打开 Plug-in Code Generators 对话框, 如图 13-2 所示。选择 Plug-in with a multi-page editor 模板, 如图 13-2 所示, 并单击 Next。

在下一个对话框中, 输入供应商的名称 “Eclipse Cookbook”, 如图 13-3 所示, 并单击 Next。

在下一个对话框中, 可以设置用于由这个插件创建的文件名的扩展名; 在此输入 `new`, 如图 13-4 所示。然后单击 Next。

在最后一个对话框中, 如图 13-5 所示, 把文件扩展名设置为 `new`, 并输入 `document.new` 作为初始文件名。

最后, 单击 Finish, 以创建插件的代码模板。下面是由 PDE 创建并添加到 `org.cookbook.ch13.EditorPlugIn` 项目中的文件。

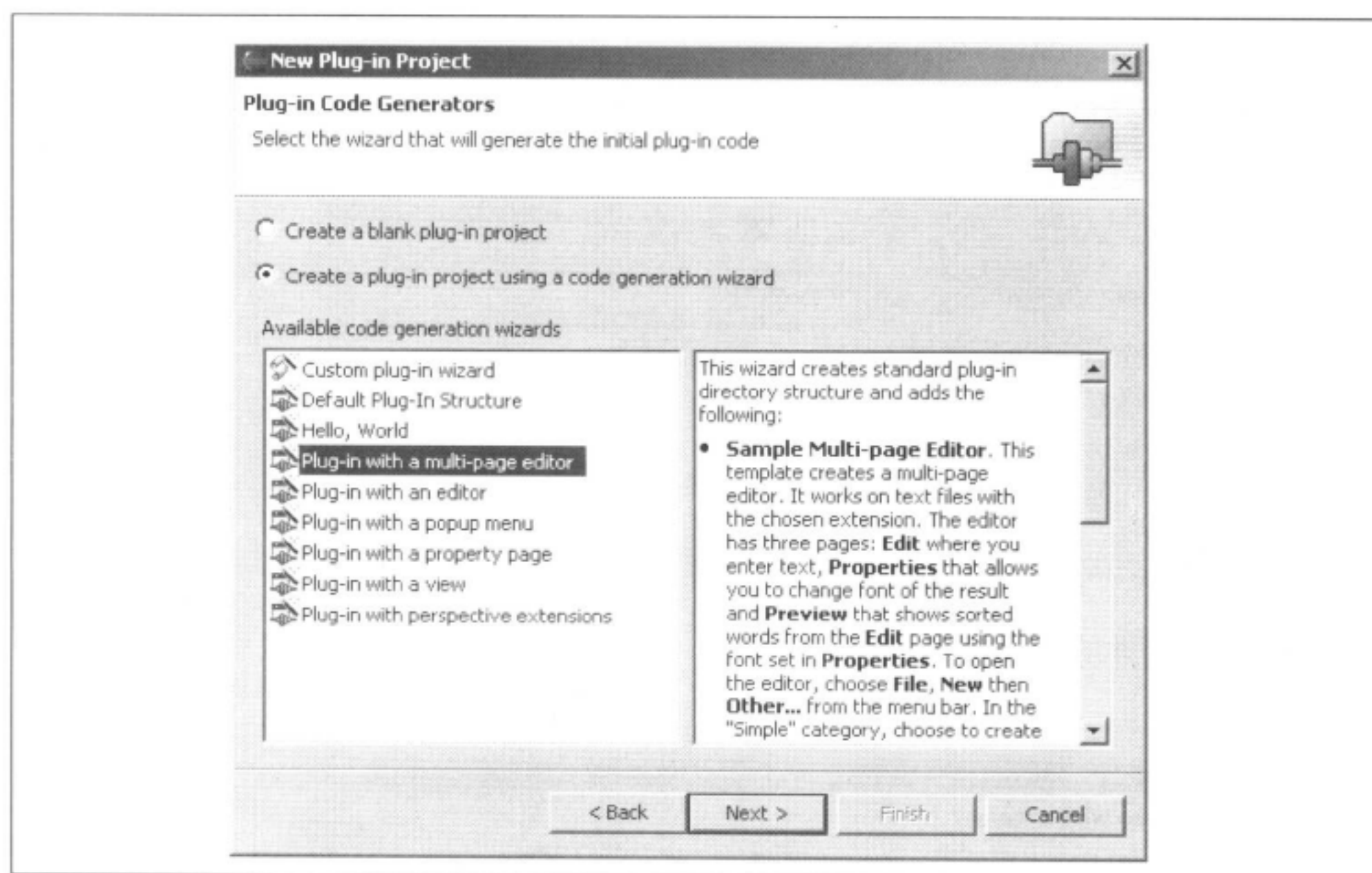


图 13-2：创建一个具有多页编辑器的插件

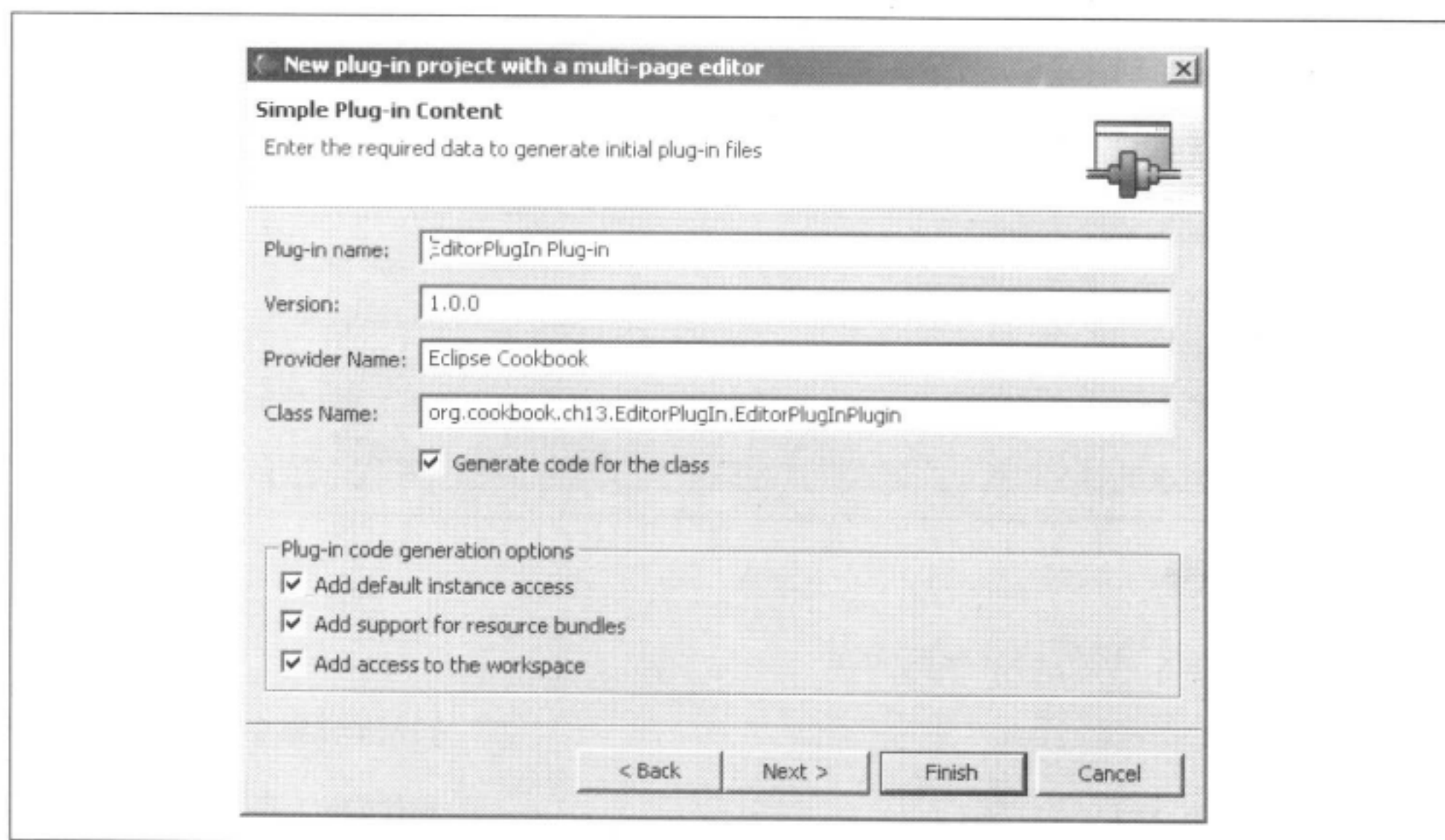


图 13-3：输入供应商的名称

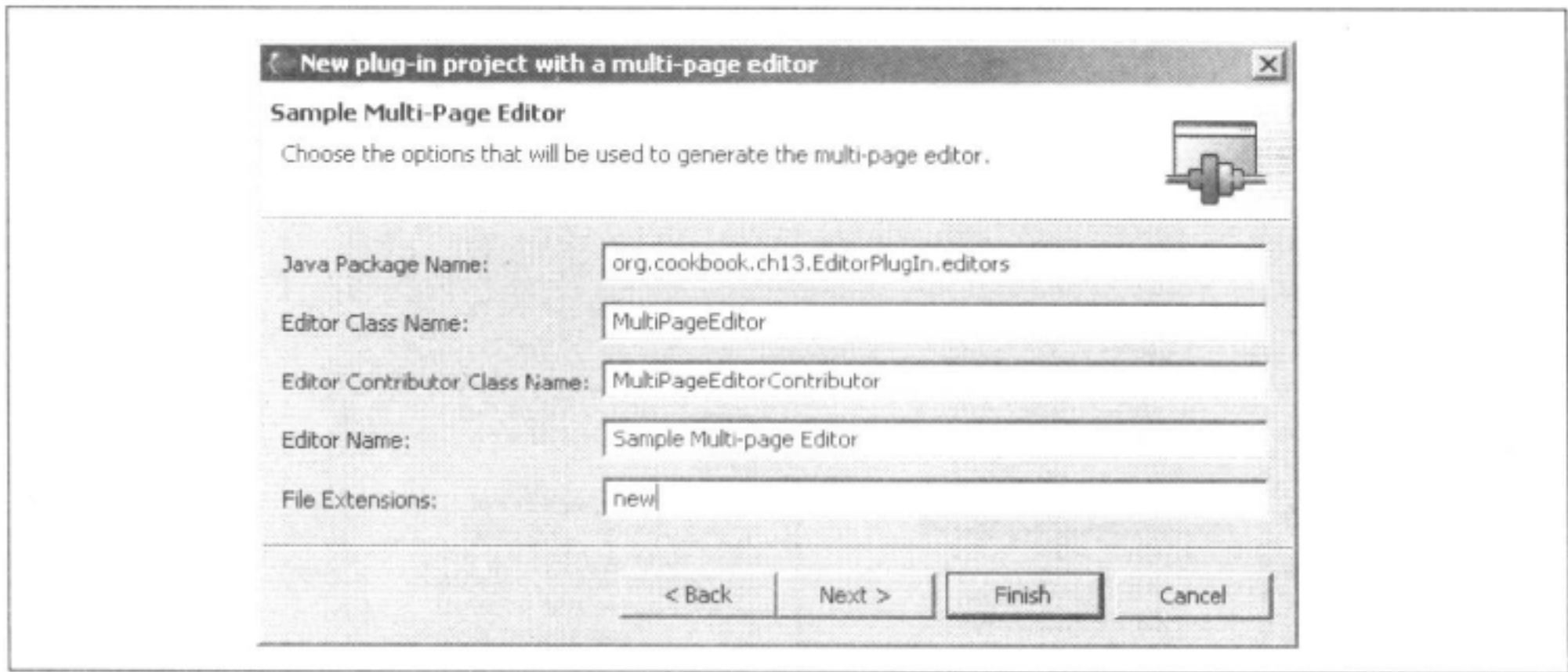


图 13-4：配置插件的编辑器

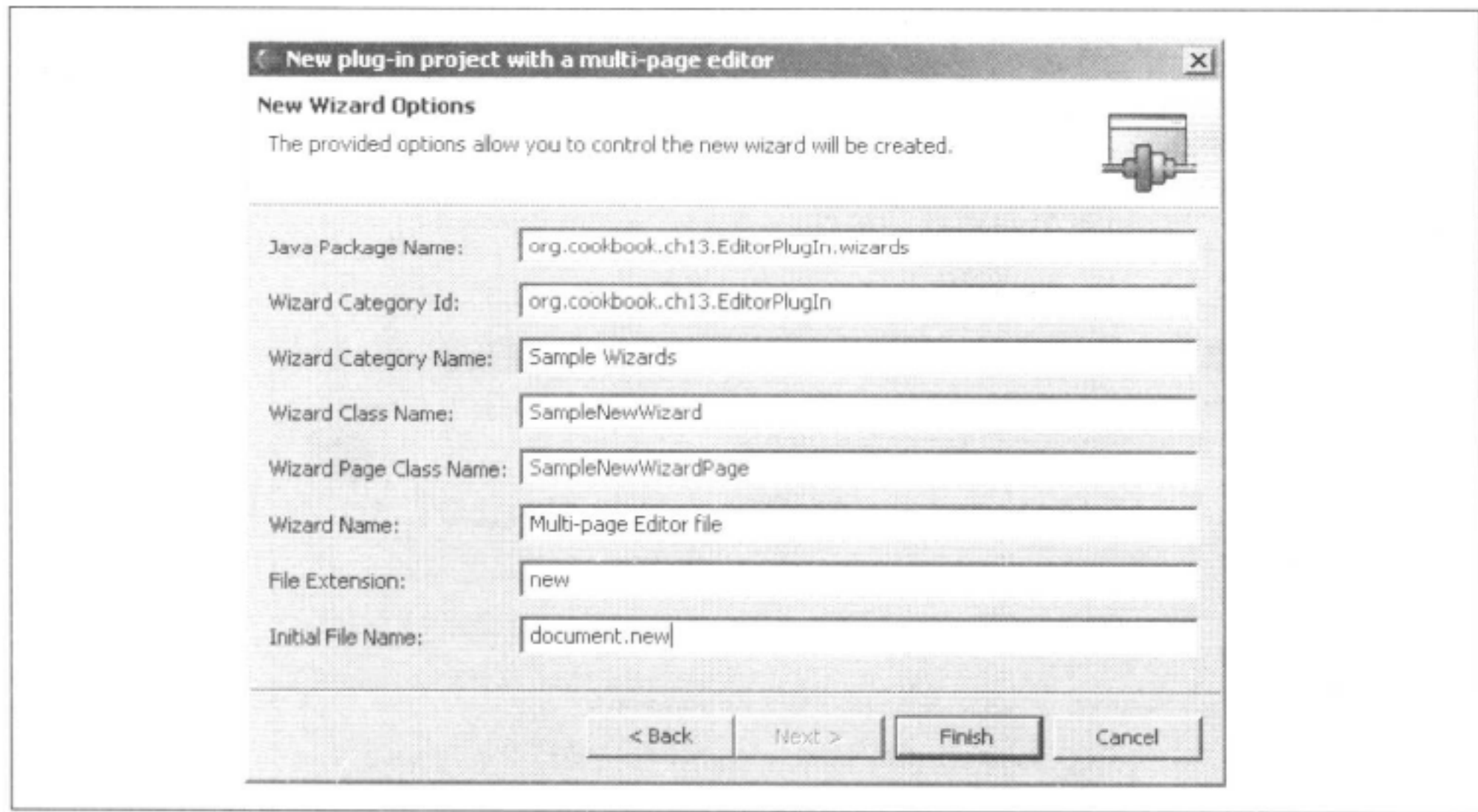


图 13-5：配置向导

```
org.cookbook.ch13.EditorPlugIn
|
|__build.properties           控制编译脚本
|
|__plugin.xml                 插件的信息清单
|
|__src                        源文件夹
|
|__org.cookbook.ch13.EditorPlugIn
```

	__EditorPlugInPlugin.java	插件的 Java 文件
	__org.cookbook.ch13.EditorPlugIn.editors	
	__MultiPageEditor.java	编辑器的代码
	__MultiPageEditorContributor.java	编辑器动作
	__org.cookbook.ch13.EditorPlugIn.wizards	
	__SampleNewWizard.java	向导的代码
	__SampleNewWizardPage.java	向导的页面

参考

13.2 节，自定义向导；13.3 节，自定义编辑器；*Eclipse* (O'Reilly) 一书的第 12 章。

13.2 自定义向导

问题

你想自定义为包含向导的插件创建的模板。

解决方案

编辑向导的支持文件（即上一节创建的示例中的 *SampleNewWizard.java* 文件），以自定义代码。

讨论

在上一节创建的插件中，向导创建默认文件名为 *document.new* 的文件，将默认的文本插入这些文件中。这个向导得到了两个文件的支持，这两个文件是 *SampleNewWizard.java* 和 *SampleNewWizardPage.java*。在文件 *SampleNewWizard.java* 中，代码首先创建向导将要显示的页面：

```
public class SampleNewWizard extends Wizard implements INewWizard {
    private SampleNewWizardPage page;
    private ISelection selection;

    /**
     * Constructor for SampleNewWizard.
     */
    public SampleNewWizard() {
        super();
        setNeedsProgressMonitor(true);
    }
}
```

```

/**
 * Adding the page to the wizard.
 */

public void addPages() {
    page = new SampleNewWizardPage(selection);
    addPage(page);
}
.
.
.

```

在 *SampleNewWizardPage.java* 的 `createControl` 方法中, 创建向导的外观。正如下面的代码所示, 在向导页面上添加 SWT 控件——标签、文本等控件, 并设定这些控件的位置:

```

public void createControl(Composite parent) {
    Composite container = new Composite(parent, SWT.NULL);
    GridLayout layout = new GridLayout();
    container.setLayout(layout);
    layout.numColumns = 3;
    layout.verticalSpacing = 9;
    Label label = new Label(container, SWT.NULL);
    label.setText("&Container:");

    containerText = new Text(container, SWT.BORDER | SWT.SINGLE);
    GridData gd = new GridData(GridData.FILL_HORIZONTAL);
    containerText.setLayoutData(gd);
    containerText.addModifyListener(new ModifyListener() {
        public void modifyText(ModifyEvent e) {
            dialogChanged();
        }
    });

    Button button = new Button(container, SWT.PUSH);
    button.setText("Browse...");
    button.addSelectionListener(new SelectionAdapter() {
        public void widgetSelected(SelectionEvent e) {
            handleBrowse();
        }
    });

    label = new Label(container, SWT.NULL);
    label.setText("&File name:");

    fileText = new Text(container, SWT.BORDER | SWT.SINGLE);
    gd = new GridData(GridData.FILL_HORIZONTAL);
    fileText.setLayoutData(gd);
    fileText.addModifyListener(new ModifyListener() {
        public void modifyText(ModifyEvent e) {
            dialogChanged();
        }
    });
    initialize();
}

```

```

        dialogChanged();
        setControl(container);
    }

```

如果你想自定义向导的页面，就在上述代码中进行。这就是创建向导页面的代码；如果你想要一个不同的外观和不同的控件，就根据需要在这里进行配置。

向导为插件所做的实际工作是从在向导页面上单击 Finish 按钮开始的，单击 Finish 按钮时将调用 *SampleNewWizard.java* 文件中的 *doFinish* 方法。我们将自定义这个方法的代码，以便将我们的文本插入到新建的文档中。这个方法在一个工人线程中运行，并首先找到要创建的文件的路径，并以这种方式创建文件；如果需要在另一个位置创建文件，就在这里进行自定义：

```

private void doFinish(
    String containerName,
    String fileName,
    IProgressMonitor monitor)
    throws CoreException {
    // create a sample file
    monitor.beginTask("Creating " + fileName, 2);
    IWorkspaceRoot root = ResourcesPlugin.getWorkspace().getRoot();
    IResource resource = root.findMember(new Path(containerName));
    if (!resource.exists() || !(resource instanceof IContainer)) {
        throwCoreException("Container \"" + containerName +
            "\" does not exist.");
    }
    IContainer container = (IContainer) resource;
    final IFile file = container.getFile(new Path(fileName));
    .
    .
    .

```

为了设置新建文件的内容，代码下一步调用一个名为 *openContentStream* 的方法，并使用这个方法填充文件：

```

try {
    InputStream stream = openContentStream();
    if (file.exists()) {
        file.setContents(stream, true, true, monitor);
    } else {
        file.create(stream, true, monitor);
    }
    stream.close();
}

```



下面我们将自定义 *openContentStream* 方法，以修改填充到向导创建的 *document.new* 文件中的默认文本。这个方法的代码如下：

```
private InputStream openContentStream() {
    String contents =
        "This is the initial file contents for *.new file that should
        be word-sorted in the Preview page of the multi-page editor";
    return new ByteArrayInputStream(contents.getBytes());
}
```

下面是修改后的代码，其中添加了我们自定义的文本：

```
private InputStream openContentStream() {
    String contents =
        "Welcome to document.new";
    return new ByteArrayInputStream(contents.getBytes());
}
```

进行上述修改后，就按照我们的意图自定义了向导。在 13.3 节中，我们将自定义这个插件显示的编辑器。

参考

13.1 节，创建支持向导和编辑器的插件；13.3 节，自定义编辑器。

13.3 自定义编辑器

问题

你已经使用 PDE 向导创建了一个新的插件，而你想自定义插件的编辑器。

解决方案

自定义编辑器的 *.java* 文件。

讨论

如果你一直在跟着前两节的讨论进行学习，那么现在应该已经有了一个不错的、用于带编辑器的插件的模板，而自定义这个模板是一件简单的事情。这个编辑器需要两个文件的支持，这两个文件是 *MultiPageEditor.java* 和 *MultiPageEditorContributor.java*。*MultiPageEditorContributor.java* 文件提供工具栏和菜单支持，而 *MultiPageEditor.java* 文件为编辑器提供实际的 Java 支持。

在这个例子中，我们要查看一下 *MultiPageEditor.java* 的代码，并对它进行修改。按照现在的情况，代码将显示 3 个页面：一页显示插件向导创建的文档中的文本，一页使你

能够选择显示的字体，而另一页按顺序显示文档中的单词。在本节的例子中，不需要显示所有这3页，只是想利用第一页在一个文本编辑器中显示文档中的文本。

要在编辑器的第一页中显示文本，代码需要使用一个名为 `editor` 的 `org.eclipse.editors.ui.text.TextEditor` 对象。在创建了这个对象之后，可以使用 `MultiPageEditorPart` 类的 `addPage` 方法，在编辑器中添加新页。下面的代码说明了如何添加新页；代码首先新建一个 `TextEditor` 对象（`MultiPageEditor` 类扩展了 `MultiPageEditorPart` 类，后者使用一个 SWT 标签页控件来显示多个编辑器页。

```
public class MultiPageEditor extends MultiPageEditorPart {  
  
    private TextEditor editor;  
    .  
    .  
    .  
    void createPage0() {  
        try {  
            editor = new TextEditor();  
            .  
            .  
            .  
        }  
    }  
}
```

要将这个编辑器添加到页面上，只需使用 `addPage` 方法把它传递给 `editor` 对象，并使用 `getEditorInput` 方法获得编辑器中的文本即可。要设置这个页面标签上的文本，可使用 `setPageText` 方法：

```
public class MultiPageEditor extends MultiPageEditorPart {  
  
    private TextEditor editor;  
    .  
    .  
    .  
    void createPage0() {  
        try {  
            editor = new TextEditor();  
            int index = addPage(editor, getEditorInput());  
            setPageText(index, editor.getTitle());  
        } catch (PartInitException e) {  
            ErrorDialog.openError(  
                getSite().getShell(),  
                "Error creating nested text editor",  
                null,  
                e.getStatus());  
        }  
    }  
}
```

这就是在插件页上添加一个 `editor` 对象所需要的全部代码。为了在插件页上添加自定义

义的控件，可创建一个 SWT Composite 控件，在复合控件中设置布局，把需要的控件和监听程序添加到复合控件中，并把这个复合控件传递给 `addPage` 方法。

调用各种页面创建方法的方法叫做 `createPages`。在这个例子中，我们只需要第一个编辑器页面——由 `createPage0` 方法创建，所以应注释掉 PDE 向导创建的其他两个页面：

```
protected void createPages() {  
    createPage0();  
    // createPage1();  
    // createPage2();  
}
```

要运行这个新插件，可启动 Run-time Workbench，并新建一个 Java 项目 *TestProject*。右击 *TestProject* 项目，并选择 `New → Other`，打开 New 对话框，如图 13-6 所示。在左边的列表框中选择 `Sample Wizards`，而在右边的列表框中选择 `Multi-page Editor file`，然后单击 `Next`，打开这个新的向导。

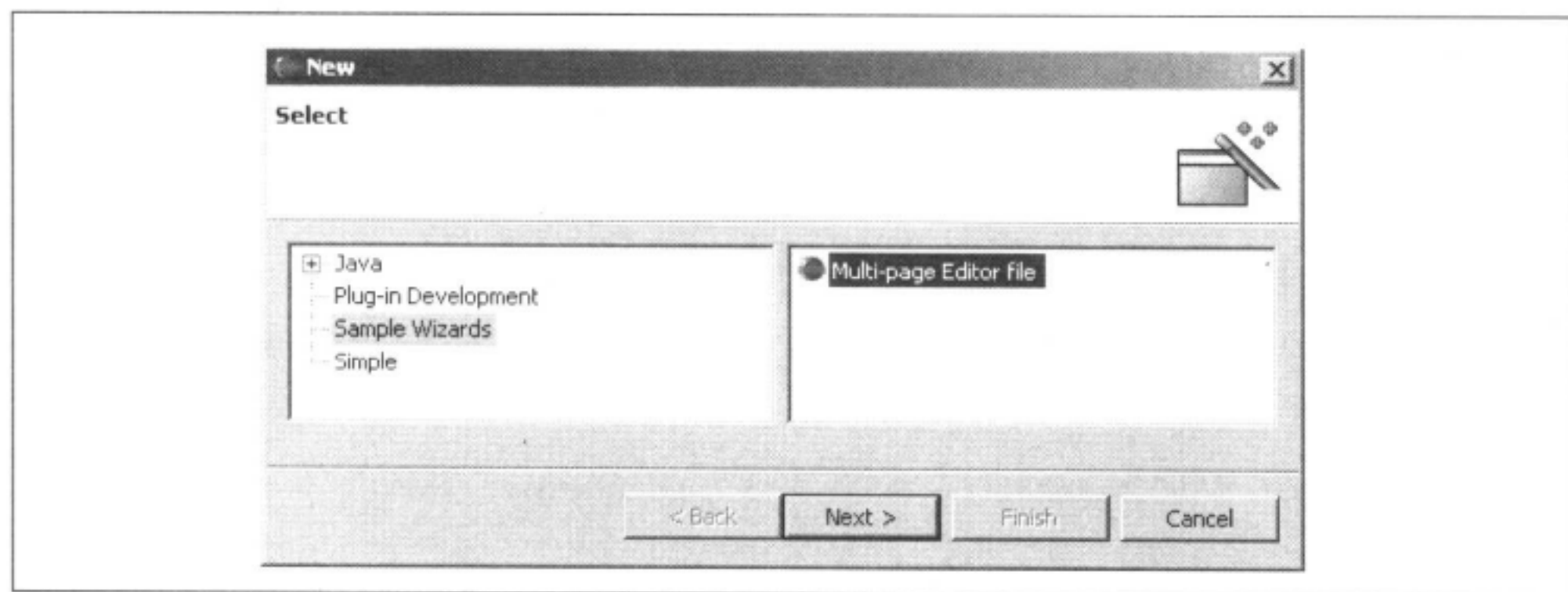


图 13-6：使用新的向导

新的向导显示，默认情况下新建文件将被命名为 *document.new*，如图 13-7 所示，但可以在此修改文件名。必须将一个项目与这个新文件相关联，所以单击 `Browse` 按钮，找到 *TestProject* 项目。

单击 `Finish`，以创建新的文档 *document.new*，此文档将被添加到指定的项目 *TestProject* 中，并在插件的编辑器中打开，如图 13-8 所示。在编辑器的标签上可以看到这个文档的名称，在编辑器中可以看到我们放置在文件中的默认文本。

如果在 Package Explorer 或 Navigator 等视图中双击一个扩展名为 *.new* 的文件，将在编辑器中自动打开该文件。可以编辑文件，并使用 Eclipse 的 `File → Save` 或 `File → Save As` 菜单项或相应的工具栏按钮保存文件。就此而言，在编辑器中添加其他页并不难；只需使用 `addPage` 方法添加该页，并用 `setPageText` 方法设置新页标签上的文本即可。

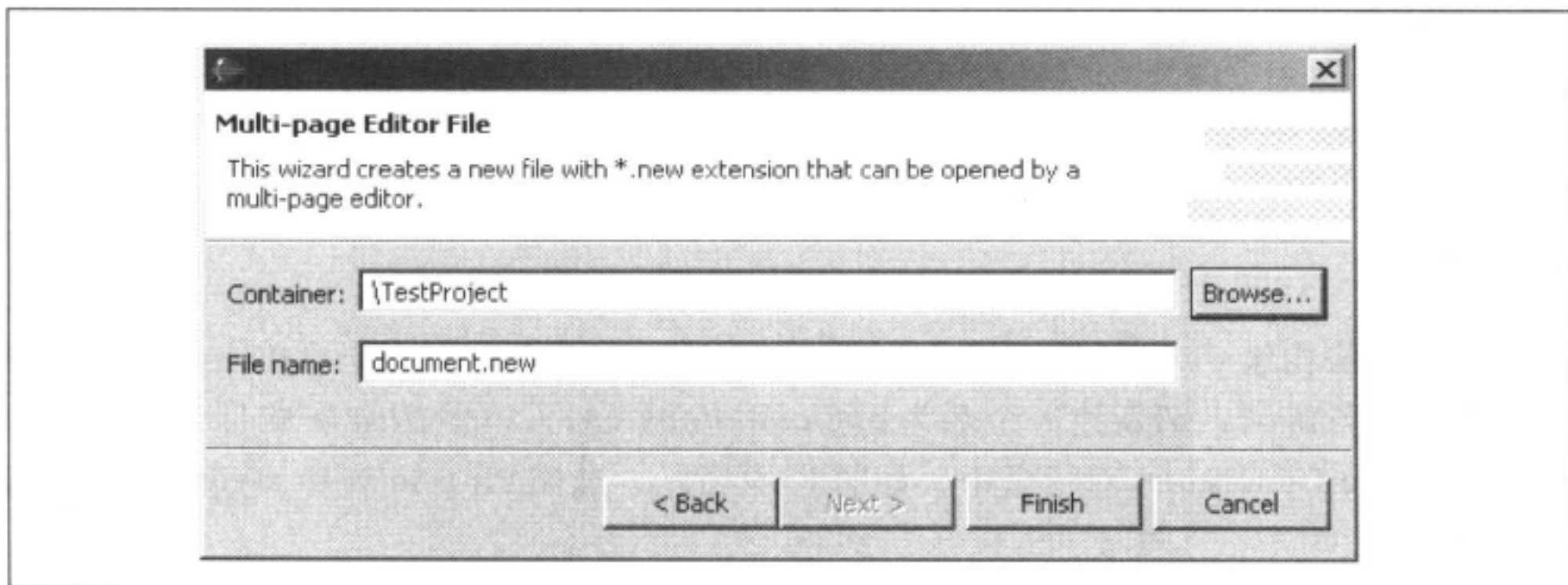


图 13-7：新建一个文档

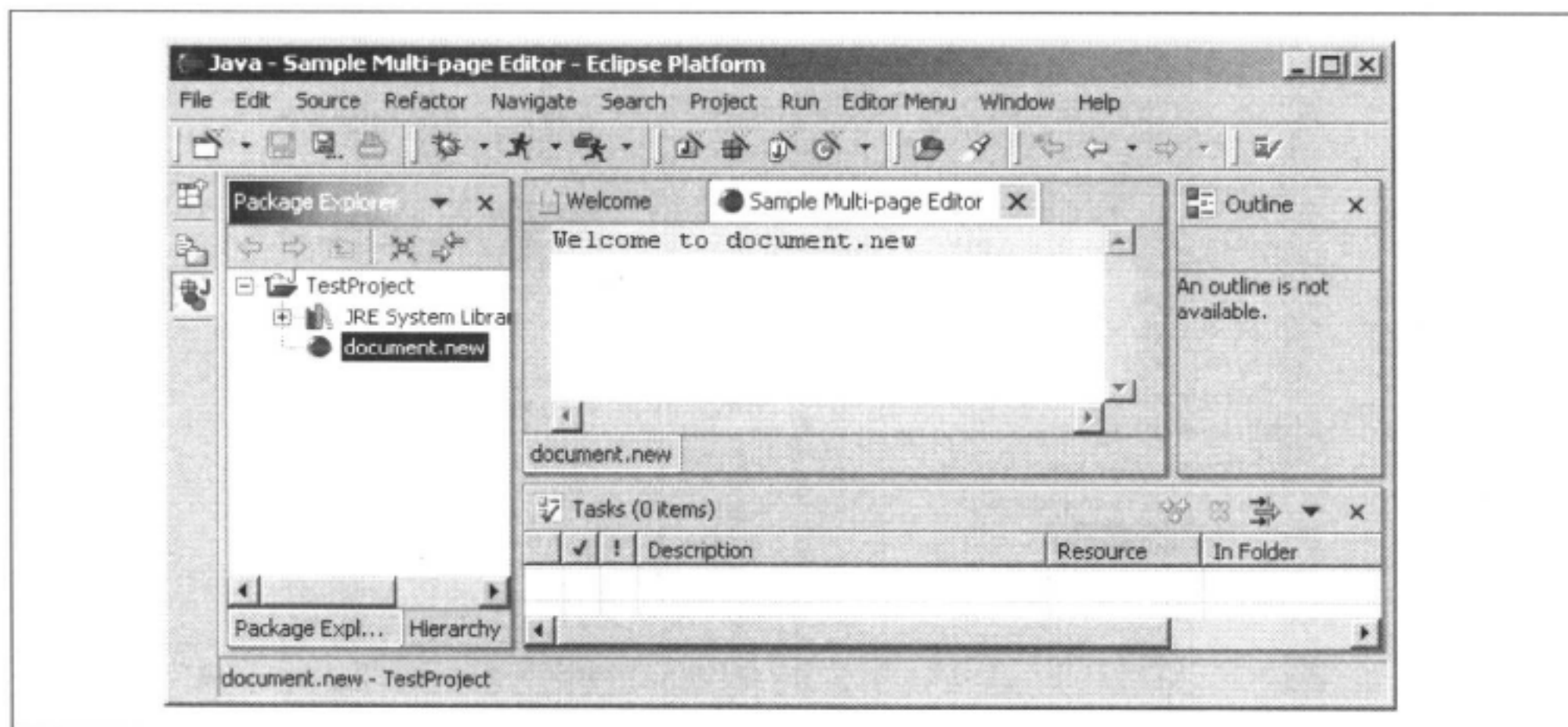


图 13-8：新建的文件

参考

13.1 节，创建支持向导和编辑器的插件；13.2 节，自定义向导；10.1 节，创建 SWT 标签文件夹；*Eclipse* (O'Reilly) 一书的第 12 章。

13.4 创建支持视图的插件

问题

你需要让插件支持视图。

解决方案

使用 PDE 模板 Plug-in with a view。

讨论

下一个例子将构建一个支持视图的插件——在这个例子中，有一个可单击的项树。要跟随本书进行练习，请新建一个名为 *org.cookbook.ch13.ViewPlugIn* 的插件项目。在 Plug-in Code Generators 对话框中，如图 13-9 所示，选择 Plug-in with a view，并单击 Next。

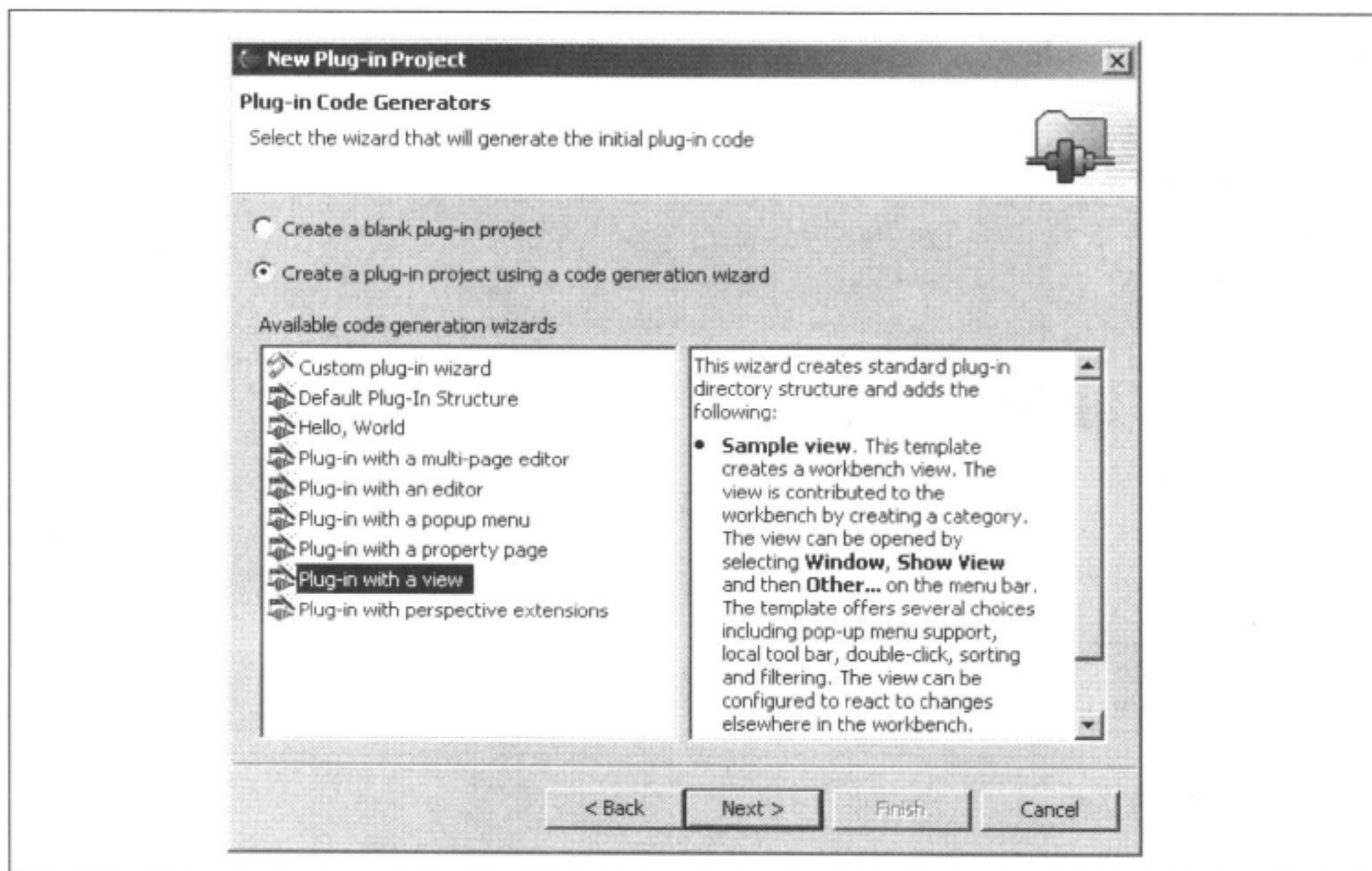


图 13-9：配置支持视图的插件

在下一个对话框中，把供应商名称设置为 Eclipse Cookbook，如图 13-10 所示。

单击 Next，打开 Main View Settings 对话框，如图 13-11 所示。在这个对话框中，可以自定义视图，设置视图的名称以及该视图是一个基于树的视图还是一个基于表格的视图。将这个视图命名为 Item View，并将其设置为一个基于树的视图，如图 13-11 所示。再次单击 Next，打开这个向导的最后一个对话框。

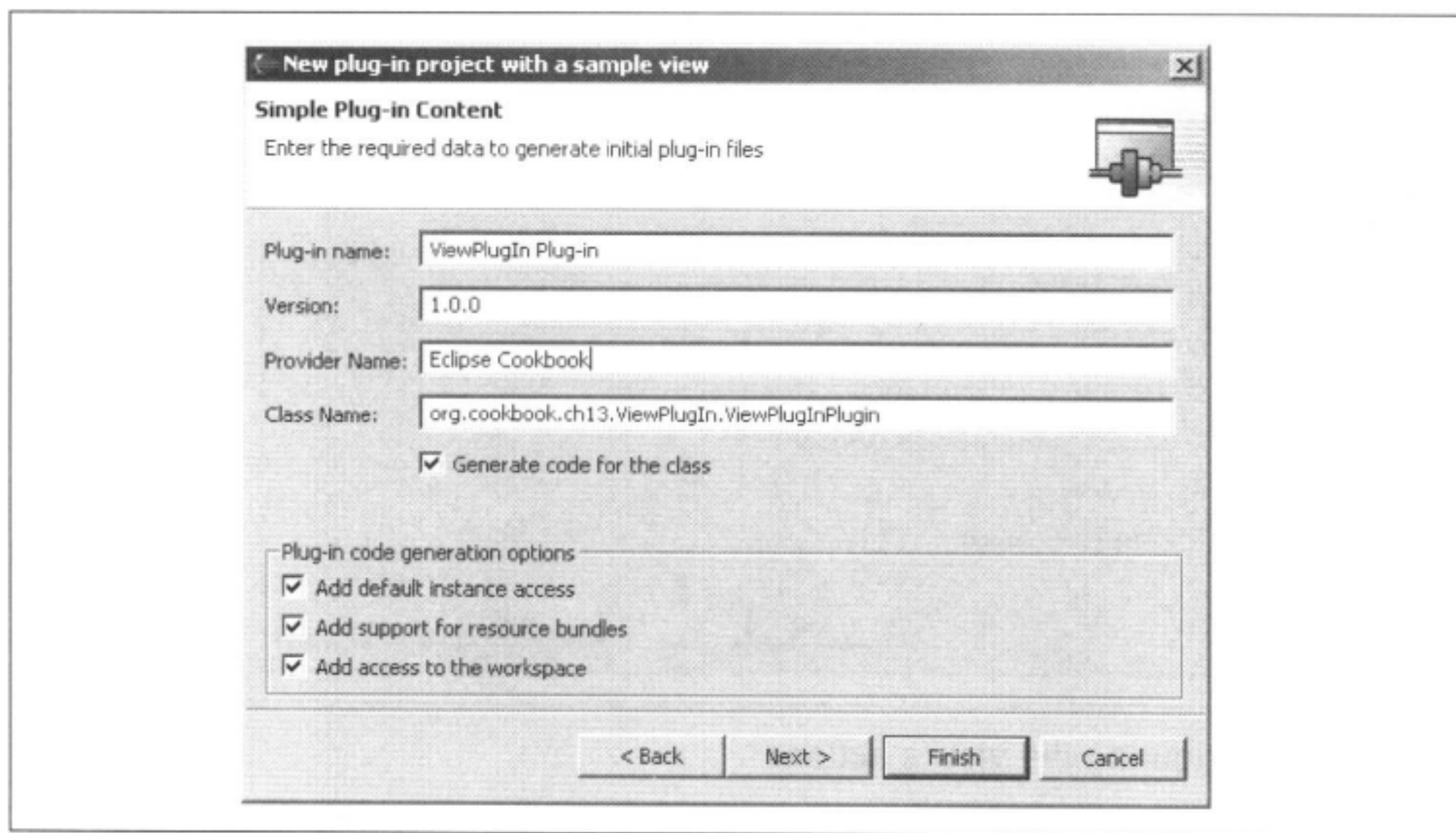


图 13-10：设置供应商名称

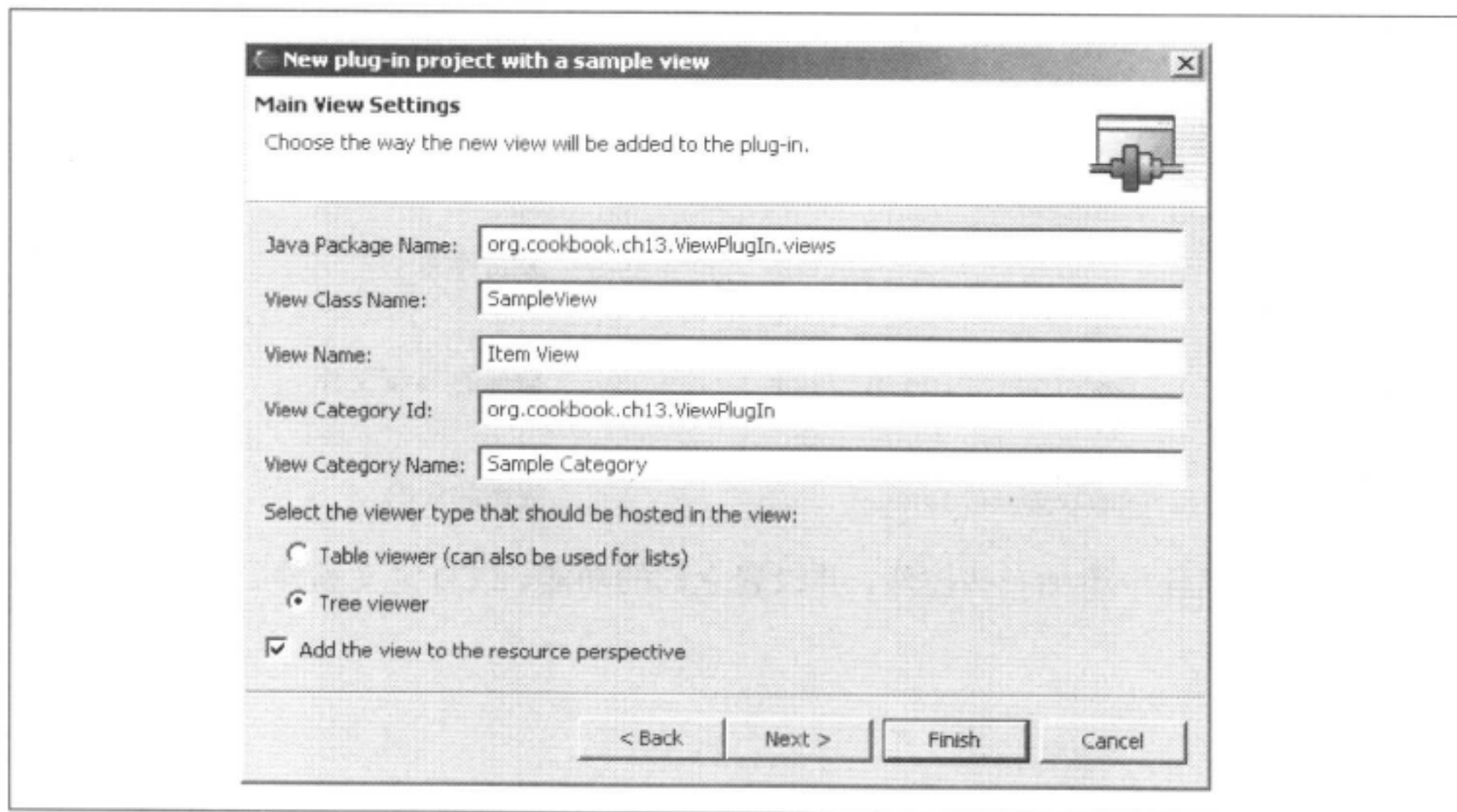


图 13-11：设置视图选项

在最后一个对话框中，可以配置视图的动作。保留默认设置，并单击 Finish，以创建这个插件的模板（参见图 13-12）。

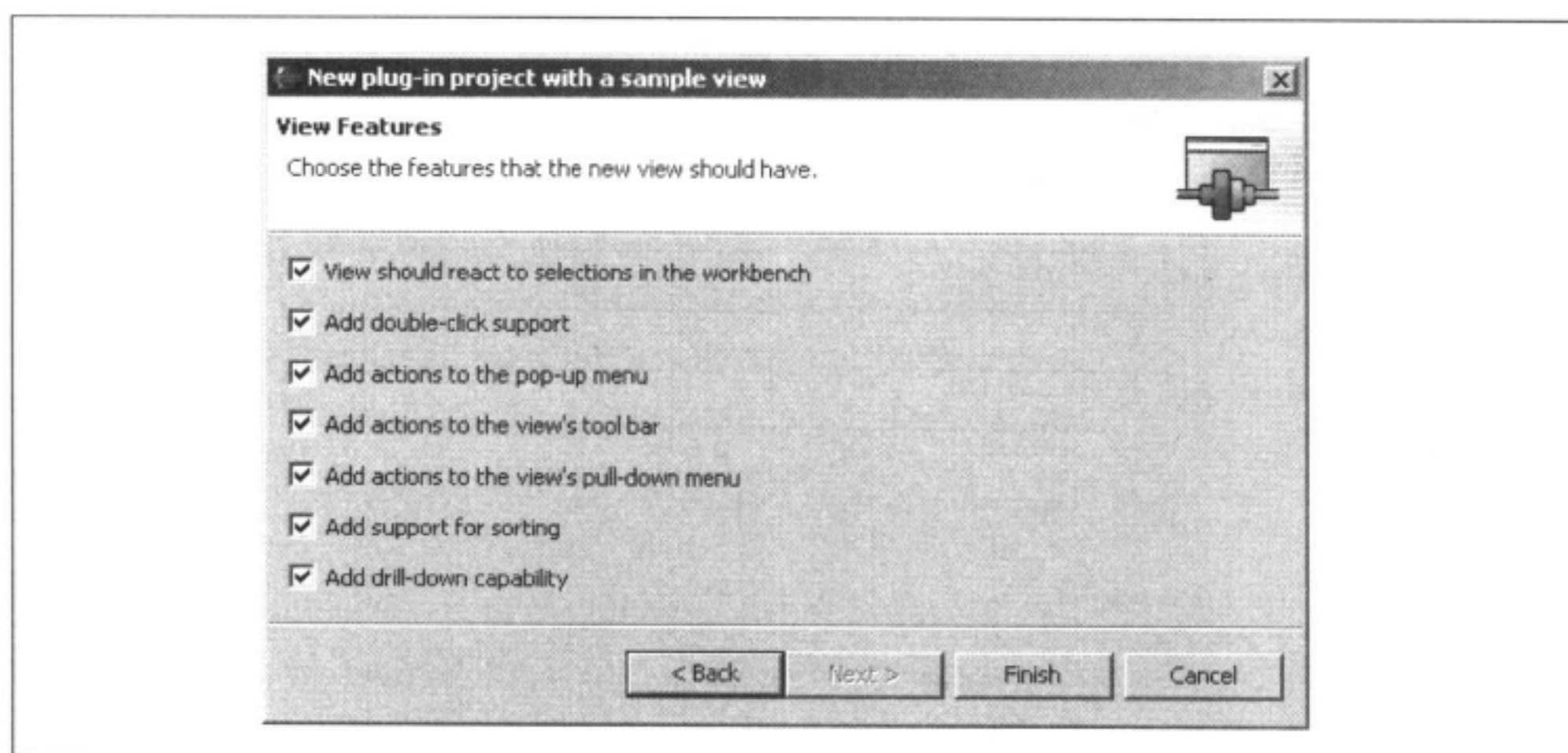


图 13-12: Configuring the view's actions

在单击 Finish 之后，即可为这个插件创建模板，并且创建下列文件，并添加到项目的 *src* 文件夹中：

org.cookbook.ch13.ViewPlugIn	
___.build.properties	控制编译脚本
___.plugin.xml	插件信息清单
___.src	源文件夹
___.org.cookbook.ch13.ViewPlugIn	
___.ViewPlugInPlugin.java	插件的 Java 文件
___.org.cookbook.ch13.ViewPlugIn.views	
___.SampleView.java	视图的代码

要在这个新的插件中添加一些控件，并设置它们的功能，以自定义插件，可参阅接下来的两节。

参考

13.5 节，在视图添加控件；13.6 节，配置视图的动作；*Eclipse* (O'Reilly) 一书的第 12 章。

13.5 在视图中添加控件

问题

你已经创建了一个支持视图的插件，而你想在该视图中添加一些控件。

解决方案

如果你使用一个 PDE 模板创建了该插件，只需自定义 `initialize` 方法即可。

讨论

如何自定义视图中控件，取决于该视图是基于 SWT 表格还是基于 SWT 树。在上一节开发的例子中，视图 *ViewPlugIn* 是基于树的。可以在 *SampleView.java* 的 `initialize` 方法中，在一个名为 `invisibleRoot` 的 `TreeParent` 对象下，创建树结构。下面的代码在视图中创建一个节点树（注意，在真实的应用程序中，这个树应反映具体的数据模型）。

```
private void initialize() {
    TreeObject to1 = new TreeObject("Item 1");
    TreeObject to2 = new TreeObject("Item 2");
    TreeObject to3 = new TreeObject("Item 3");
    TreeObject to4 = new TreeObject("Item 4");
    TreeParent p1 = new TreeParent("Parent 1");
    p1.addChild(to1);
    p1.addChild(to2);
    p1.addChild(to3);
    p1.addChild(to4);

    TreeObject to5 = new TreeObject("Item 5");
    TreeParent p2 = new TreeParent("Parent 2");
    p2.addChild(to4);

    TreeParent root = new TreeParent("Root");
    root.addChild(p1);
    root.addChild(p2);

    invisibleRoot = new TreeParent("");
    invisibleRoot.addChild(root);
}
```

上述代码配置了将在视图中出现的树对象。要使这些控件在单击（用任何一个鼠标键）或双击时能够实际实现某种功能，请参阅下一节。

参考

13.4 节，创建支持视图的插件；13.6 节，配置视图的动作；*Eclipse* (O'Reilly) 一书的第 11 章和第 12 章。

13.6 配置视图的动作

问题

你需要使视图中的控件在单击、右击或双击时发挥作用。

解决方案

配置视图的动作。如果是使用一个 PDE 模板创建的插件，编辑 `makeActions` 方法。

讨论

在前两节中，我们创建了一个基于 SWT 树控件的、支持视图的插件。要使上一节中添加到视图中的控件在与用户交互时发挥作用，可编辑 `SampleView.java` 中的 `makeActions` 方法的代码。在这个方法的代码中，可以使用下列这行代码访问被单击、右击或双击的控件：

```
Object obj = ((IStructuredSelection)selection).getFirstElement()
```

下面的代码说明了如何修改 `makeActions` 方法的代码，以便创建动作，来处理单击或双击视图中的控件时的菜单选择：

```
private void makeActions() {
    action1 = new Action() {
        public void run() {
            showMessage("You selected Action 1");
        }
    };
    action1.setText("Action 1");
    action1.setToolTipText("Action 1 tooltip");
    action1.setImageDescriptor(PlatformUI.getWorkbench().getSharedImages().
        getImageDescriptor(ISharedImages.IMG_OBJS_INFO_TSK));

    action2 = new Action() {
        public void run() {
            showMessage("You selected Action 2");
        }
    };
    action2.setText("Action 2");
    action2.setToolTipText("Action 2 tooltip");
}
```

```
        action2.setImageDescriptor(PlatformUI.getWorkbench().getSharedImages().  
            getImageDescriptor(ISharedImages.IMG_OBJ_TASK));  
        doubleClickAction = new Action() {  
            public void run() {  
                ISelection selection = viewer.getSelection();  
                Object obj =  
                    ((IStructuredSelection)selection).getFirstElement();  
                showMessage("You double-clicked " + obj.toString());  
            }  
        };  
    }  
}
```

要使用 Item View 视图，可启动 Run-time Workbench，并选择 Window → Show View → Other。在 Sample Category 文件夹中选择 Item View，如图 13-13 所示，并单击 OK，以显示新的视图。

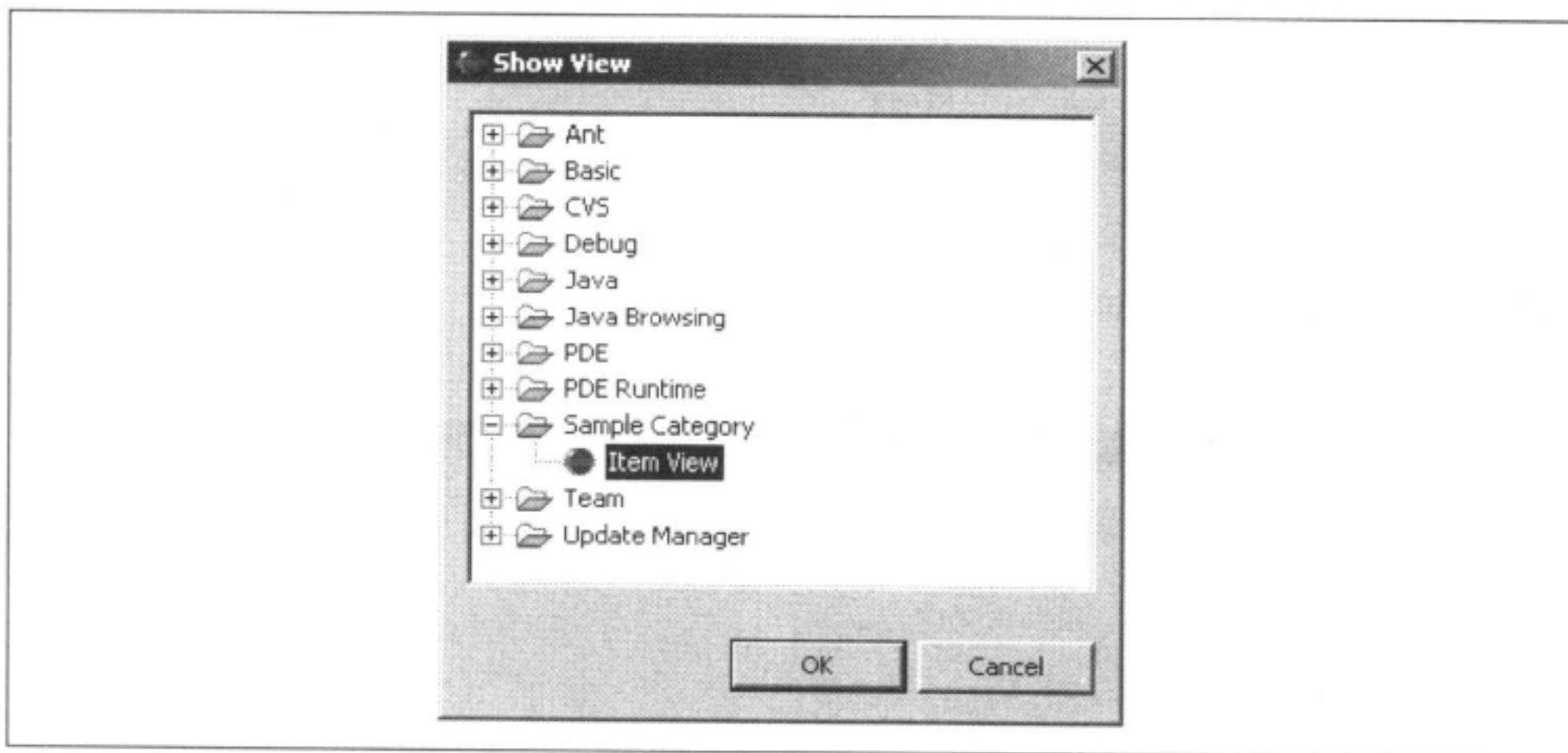


图 13-13：选择 Item View

新建的视图出现在图 13-14 的底部。这是一个新的、有效的、由我们添加到 Eclipse IDE 的视图。

双击视图中的一个控件可以激活与双击相关联的动作，如图 13-15 所示。

就此而言，也可以右击一个控件，并从出现的快捷菜单中选择 Action 1 或 Action 2。一个消息框可以显示你选择了哪个动作，如图 13-16 所示。

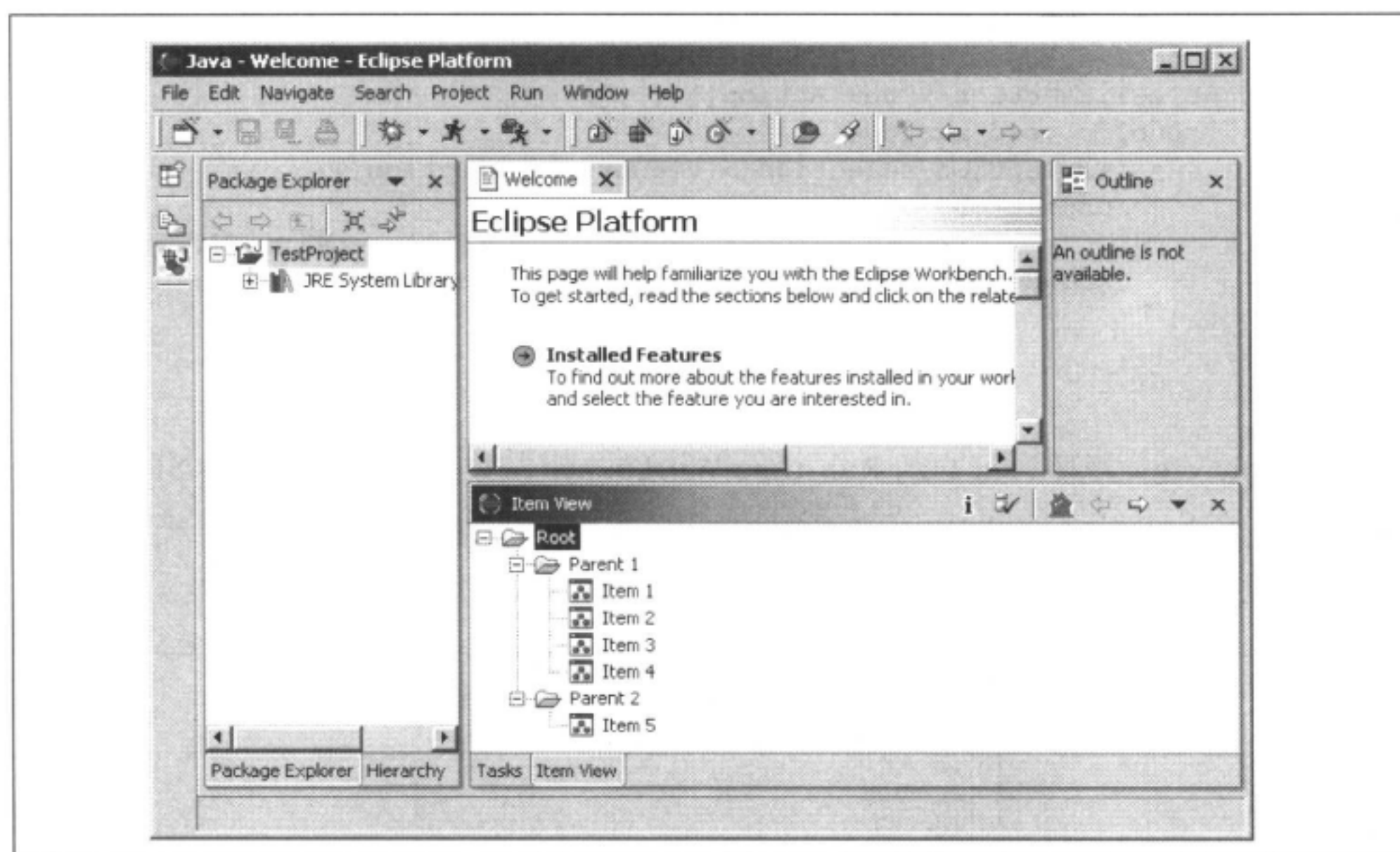


图 13-14: 新建的 Item View 视图

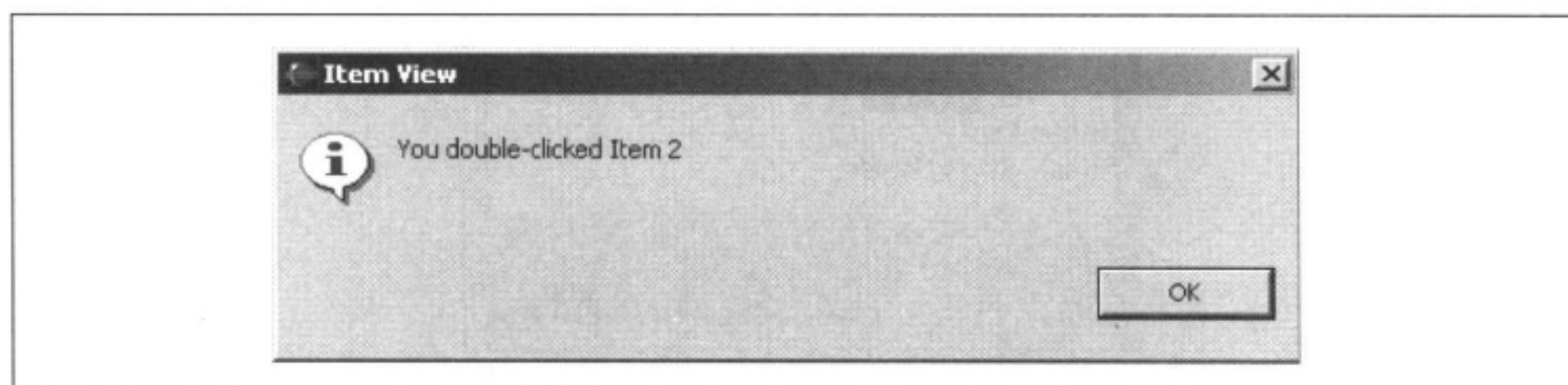


图 13-15: 双击一个控件

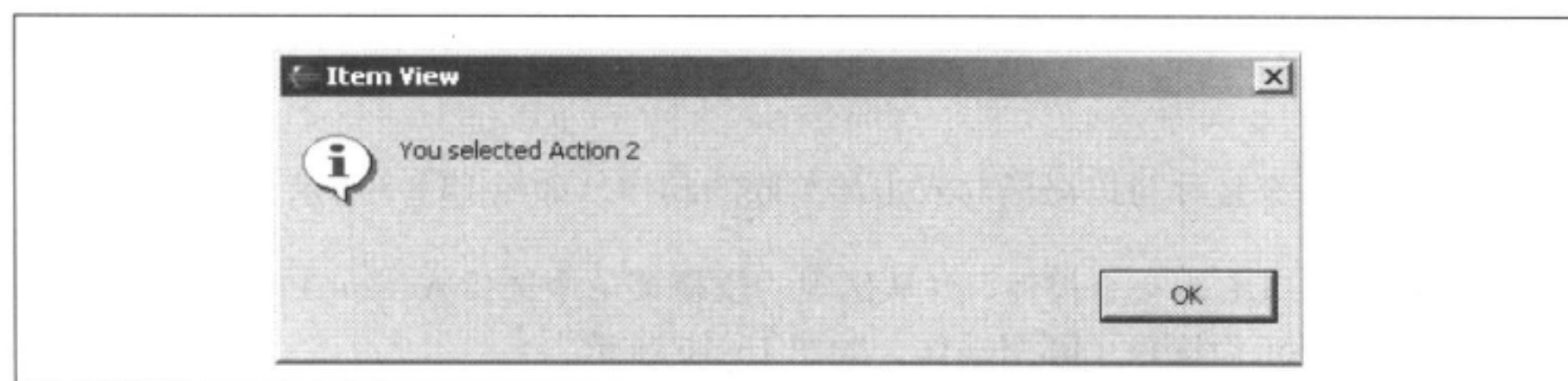


图 13-16: 选择一个快捷菜单项

这表明我们的视图插件是可用的。显然，在 Eclipse 中添加一个视图毕竟不是太难。可以使用 PDE 工具创建一个模板，并修改模板的代码。并不限制视图是基于表格还是树（虽然这是传统的做法），而且可以在视图中使用 SWT 复合控件来容纳自定义控件。

参考

13.4 节，创建支持视图的插件；13.5 节，在视图添加控件；*Eclipse* (O'Reilly) 一书的第 11 章和第 12 章。